# Software Project Staffing: From empirical and analytical perspectives

A Dissertation Proposal

Submitted to the Faculty

of

Purdue University

by

Keumseok Kang

October 2009

# TABLE OF CONTENTS

# LSIT OF TABLES

# LIST OF FIGURES

## Abstract

Developing software for organizations is a very complex activity because it must deal with people, organizations, technologies, and business processes. Despite recent advances in technology and management practices, the software organizations still suffer from difficulties in software development management.

One of the difficulties in software development management lies in the staffing problems. Software development is an interdependent group task. A group of people with different knowledge and skills, which we call a software project team, work together to develop software. Accordingly, the project team influences the outcome of software development. Therefore, project staffing, that is, how to form software project teams, has persistently been a key question of software organizations. We investigate software project staffing problems in this research.

Our research consists of three studies. In the first study, we investigate what type of knowledge, among domain, technology, and methodology knowledge, is most influential to the performance of software development. We empirically 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.

In the second study, we investigate which professional type, among specialists, those with deep / focused experiences within a particular domain, and generalists, who have experiences with a broad range of domains, is more valuable to software projects. We empirically find that the impact of the type of professionals' experiences (i.e., specialist vs. generalist) varies by the type of software development knowledge required (i.e., domain, technology, and methodology) and by role of the software professional within the project (i.e., project managers vs. non-project managers).

In the third study, we solve the software project selection problem of an IT service company. We define the problem using a new version of secretary problem, which we call the *stochastic multiple-secretary problem with deferred decisions,* and find a threshold type optimal policy for this problem.

Our findings and solutions provide managerial implications not only to project staffing problems but also to other organizational issues such as knowledge development, career development, and project selection. In addition, both empirical and analytical perspectives enable us to provide a full-fledged solution to the project staffing problem.

# CHAPTER 1. 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 management and engineering has been regarded as one of the most complex problem domains and has been studied for decades. However, recent statistics imply that the software industry still suffers from high failure rates and poor 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 overruns of 56% and average schedule overruns of 84%.

Software development is an interdependent group task. A group of people with different knowledge and skills, which we call a software project team, works together to develop software. A software project team is a temporal organization, which is formed when a new software development project is initiated and dismissed when the development is completed. Accordingly, the project team influences the outcome of software development. Therefore, project staffing, that is, how to form software project teams, has persistently been a key question of software organizations.

We investigate project staffing problems in this research, which consists of three studies. In the first study, we investigate learning and forgetting curve effects in software development. Previous literature shows that learning curves exist in software development (Boh, Slaughter, & Espinosa 2007; Huckman, Staats, & Upton 2009; Langer, Slaughter, & Mukhopadhyay 2008; Narayanan, Balasubramanian, & Swaminathan 2009). As an extension, we first hypothesize that forgetting curves also exist in software development. In addition, using cognitive theories, we argue that learning curves and forgetting curves vary by the type of knowledge and skills required – domain, technology, and methodology knowledge. In the second study, we investigate what type of experience – specialized vs. diverse experience – is more needed in software development. Recent literature implies that both specialization and diversity of

experience can enhance the performance of software development (Boh et al. 2007; Narayanan et al., 2009). As an extension, we argue that the most needed experience type varies by the type of required knowledge and project roles – project managers (PMs) vs. non project managers (non-PMs). Based on the results, we define the most appropriate professional type – specialist vs. generalist vs. T-shaped professional – for software development projects. Software organizations usually conduct multiple software projects with capacity constrained human resources. Therefore, selecting software projects is a critical decision-making problem for software organizations. In the third study, we analytically investigate this selection problem, which we call the project selection problem.

Our findings and solutions provide managerial implications not only to project staffing problems but also to other organizational issues such as knowledge development, career development, and project selection. In addition, both empirical and analytical perspectives enable us to provide a full-fledged solution to project staffing problems.

# CHAPTER 2. Differential Learning and Forgetting Curves: Does Type of Knowledge Matter?

## 2.1. Introduction

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.

Because knowledge plays a key role in software development, understanding how knowledge processes impact software development performance becomes very critical to software organizations. Learning curve theory is an appropriate tool to examine this knowledge development process.

The core mechanism of learning curve theory is that performance gains materialize with repeated application of the same unit task. However, 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; Huckman et al. 2009; Langer et al. 2008; Narayanan et al. 2009), even if the projects are not exactly the same. 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.

First, 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.

Second, 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 only in similar projects continuously but should be assigned to

*unrelated* projects and exposed to various software development knowledge and skills, which may facilitate the knowledge depreciation. In other words, current practices for project staffing have much room and potential for 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. Also, we additionally examine the effects of the diversity of team members' experience levels.

Since learning and forgetting effects as well as diversity effects are evidenced as performance improvements and degradation, understanding them 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.

## 2.2. Theory and Hypothesis

### 2.2.1. 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 2-1), 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 2-1. 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.

### 2.2.2. 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 domains (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, 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.*

### 2.2.3. Differential Learning Curves: 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.

Business rule or business process, which is a kind of domain knowledge, can be a procedural knowledge in that domain context; however, when it comes to developing software systems for that domain, it is a specification of the systems and a declarative knowledge because it does not describe anything about how to conduct the software 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 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 can be argued that technology knowledge resides in between domain knowledge and methodology knowledge along the declarative – procedural knowledge continuum.[2]

[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).

[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

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.*

### 2.2.4. 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

---

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.

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).

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, 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.*

[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.

**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 technology.*

**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 methodology.*

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.

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.

## 2.3. Methodology

### 2.3.1. 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.)

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.

### 2.3.2. Measures

#### 2.3.2.1. 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]

#### 2.3.2.2. Classifications of Domain, Technology, and Methodology

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).

[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.

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 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.

#### 2.3.2.3. Independent Variables

Using the categorization scheme described above, we computed measures for amount of prior experience (*Experience*) and lapse since prior experience (*Lapse*).

##### *2.3.2.3.1. Experience variables*

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*). 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 normalized by the number of knowledge and skills as the project team-level experience.

$$Experience = \left( \frac{\sum_{i=1}^{N} \sum_{j=1}^{P_i} k_{ij}}{N \times M} \right)$$

$N$ is the number of employees in the focal project and $M$ is the number of knowledge units required for the focal project. $P_i$ represents the number of prior projects in which member $i$ participated, and $k_{ij}$ ($\leq M$) is the number of knowledge units which project member $i$ experienced in prior project $j$, among the whole knowledge units required for the focal project.

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 2 projects, which used $K_{M1}$ and $K_{M2}$, and 1 project, which used $K_{M2}$ and $K_{M3}$ ,and member $M_2$ was previously assigned to 1 project which only used $K_{M1}$, then the total number of projects experienced the methodologies either $K_{M2}$ or $K_{M3}$ by the project team is 6(=2*2+1+1) and the average number per team member is 3(=6/2). 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 1.5.

#### *2.3.2.3.2. Lapse variables*

We also computed the time interval (in days) between the start of the focal project and the end of prior project in which the same domain knowledge, same technology knowledge or same methodology

knowledge was used to measure the three *Lapse* variables. For each team member assigned to the focal project, we calculate the average time intervals between the focal project and the prior projects he/she participated in that used the same domain knowledge (*LapseDomain*), same technology knowledge (*LapseTechnology*) or same methodology knowledge (*LapseMethodology*).

$$Lapse = \left( \frac{\sum_{i=1}^{N} \sum_{j=1}^{P_i} k_{ij} \times (start\ working\ date\ in\ the\ focal\ project - end\ working\ date\ in\ project\ j)}{\sum_{i=1}^{N} \sum_{j=1}^{P_i} k_{ij}} \right)$$

The *start working date* and *end working date* in the above formula represent the date when member *i* is assigned to the focal project and the date when a member is released from project *j*, respectively.

Suppose there are two members ($M_1$ and $M_2$) in a project team who were previously (and perhaps independently) assigned to other projects that for the same domain as that of the current project. If $M_1$ experienced one 60 days ago and another 110 days ago, while $M_2$ did once 40 days ago, then the average interval of repeated domain experiences (*LapseDomain*) becomes 70 (= (60+110+40)÷3) days.

#### 2.3.2.4. Control Variables

##### *2.3.2.4.1. Project size*

We used project revenue to control for project size. 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.

##### *2.3.2.4.2. 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.

#### *2.3.2.4.3. 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.

#### *2.3.2.4.4. 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 2-2 summarizes the descriptive statistics and inter-correlations among variables.

**Table 2-2. Descriptive Statistics and Inter-correlations**

| Variables | 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**: * $p < 0.1$, ** $p < 0.05$, *** $p < 0.01$

### 2.3.3. Analytical Approach

The most common way to model learning curves is to use power functions. 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 $\beta$ is the learning rate parameter – $\beta$ becomes negative (i.e., decreasing the cost per unit), when learning curve effects exist. Larger negative $\beta$ (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.

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

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

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:

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

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

The variables $y$, $x$, $c$ and $\beta$ are the same as in (eq. 1). Variable $t$ represents the time elapsed since previous trial and $\alpha$ is the forgetting rate parameter. When forgetting curve effects exist, $\alpha$ takes a value between 0 and 1. $\alpha$ =1 means no forgetting while $\alpha$=0 means no accumulation. Smaller values of $\alpha$ 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**:

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

### 2.3.4. 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) and conditional indice (CI) are within recommended limits.

## 2.4. 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*; Model 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 2-3. 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: $\Delta R^2 = 0.027$).

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*: $\beta_6 = -0.187$, $p < 0.1$).

**Table 2-3. Regression Results**

| | Baseline Model | | Model 1 | | Model 2 | |
|---|---|---|---|---|---|---|
| **Variables** | **Estimate** | **Std Err** | **Estimate** | **Std Err** | **Estimate** | **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(*ExperienceDomain*) | | | -0.188*** | 0.0482 | -0.236*** | 0.0586 |
| ln(*ExperienceTech*) | | | -0.450*** | 0.1007 | -0.494*** | 0.1113 |
| ln(*ExperienceMethod*) | | | -0.389*** | 0.0770 | -0.436*** | 0.0982 |
| *LapseDomain* | | | | | 0.999 | 0.0038 |
| *LapseTech* | | | | | 0.998 | 0.0068 |
| *LapseMethod* | | | | | 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-$R^2$.

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*: $\beta_7$ = -0.188, $p < 0.01$; *ExperienceTechnology*: $\beta_8$ = -0.450, $p < 0.01$; and *ExperienceMethodology*: $\beta_9$ = -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* ($\chi^2$ = 4.17, $p <$

0.05). Similarly, the coefficient for *ExperienceTechnology* was significantly less than that of *ExperienceDomain* ($\chi^2 = 4.64$, $p < 0.05$). However, the difference between *ExperienceMethodology* and *ExperienceTechnology* was not significant ($\chi^2 = 0.16$, 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*: $\alpha = 0.981$, $p < 0.01$). However, lapses in domain experience or technology experience did not seem to degrade project performance (*LapseDomain*: $\alpha = 0.999$, ns; *LapseTechnology*: $\alpha = 0.998$, 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.

### 2.4.1. 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

[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.

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.

In order to further check the appropriateness of model specifications and variables definitions, we tested some alternatives for the models and variables.

#### 2.4.1.1. Granularity of Classifications for Domain, Technology, and Methodology Knowledge

As we discussed earlier, unlike typical contexts where identical tasks are repeated and so learning curves are usually found, in software development, there is (almost) no identical software system; only similar (or related) systems may exist. For this reason, software developers hardly repeat identical tasks and but only similar (or related) tasks even in the same domain and with the same technology and methodology.

As aforementioned, fortunately, prior literature showed that learning curves also exist even when only similar tasks are repeatedly practiced. Moreover, literature further showed that the degree of similarity (or relatedness) among repeated tasks affects the learning curves effects. For example, Schilling et al. (2003) found that prior experience on moderately related tasks produce the better learning curve rate than prior experience on highly related or unrelated tasks, and Boh et al. (2007) evidenced this phenomenon in software maintenance environment.

Given that the degree of similarity between successive tasks affects the learning curve rates, the differential learning curves of knowledge types may be affected by the way to determine the same tasks. In this section, we show how we defined the degree of similarity (i.e., the granularity of knowledge classifications) for software development knowledge and report the results of sensitivity analysis on degree levels.

We tested four different levels of granularity for technology knowledge, two for methodology knowledge, and three for domain knowledge classification. For example, for technology, the first level (level 1), the coarsest-grained classification, consists of 70 distinct technology units; the second level (level 2) has 351;

the third level (level 3) has 1,324; finally, the fourth level(level 4), the finest-grained classification, is composed of 5,332 distinct technology knowledge units. Classifications are developed and maintained by the firm using some ad-hoc standard taxonomy such as standard industries codes, project management standards by Project Management Institution, IEEE standard taxonomy for software engineering standards as well as internally developed taxonomy.

Whether two experiences are on the same knowledge is determined by these classifications. We computed the cumulative experiences on a particular knowledge category (e.g., C, Java, Banking, OOAD) by counting the prior experienced projects which used the same category. As classifications get finer-grained, the degree of similarity gets higher, more similar (or closely related) experiences are treated as the same experience, and the cumulative experience becomes smaller. On the other hand, as classifications get coarser-grained, the degree of similarity gets lower, more broadly related experiences are treated as the same experience, and the cumulative experience becomes larger.

First, we tested whether the granularity level affects the learning curve effects. To be consistent with previous literature (Schilling et al. 2003; Boh et al. 2007), we found that the learning curve rates form an inverted U curve with respect to the degree of similarity in domain and methodology (Domain: level 1= -0.096, level 2= -0.380, level 3= -0.323; Methodology: level 1= -0.696, level 2= -0.686). However, in technology, we could not find the inverted U shape (Technology: level 1= -0.366, level 2= -0.548, level 3= -0.75; level 4= -0.833) although we found the marginal increase in learning rates declines in the degree of similarity. In addition to the learning curve rates, $R^2$ values, which represent the fitness of model specification, have also inverted U curves showing that the granularity, which enables the fastest learning rate, also produces the best fitness.[6]

[6] Because the number of variables in the regression model remains same across the regression models for various granularity levels, $R^2$ can be used to compare the fitness of the regression models.

**Table 2-4. Learning Rates and $R^2$ of Each Granularity Level**

| | Technology | | | | Domain | | | Methodology | |
|---|---|---|---|---|---|---|---|---|---|
| Level | Level 1 | Level 2 | Level 3 | Level 4 | Level 1 | Level 2 | Level 3 | Level 1 | Level 2 |
| # of categories | 70 | 351 | 1324 | 5332 | 4 | 55 | 93 | 64 | 107 |
| Learning rate | -0.37*** | -0.55*** | -0.75*** | **-0.83***** | -0.10* | -0.38** | -0.32*** | **-0.70***** | -0.69*** |
| $R^2$ | 0.697 | 0.713 | 0.725 | **0.726** | 0.672 | 0.706 | 0.686 | **0.725** | 0.725 |

We also tested all 24 (= 4×2×3) combinations of granularity levels of domain, technology, and methodology and found that the combination consisting of the best granularity in each knowledge type still produces the best learning curves and the best model fitness. The best granularity level means the level which returns the best learning rate as well as the best model fitness: level 2 for domain, level 4 for technology, and level 1 for methodology.

We used the best granularity levels for knowledge types in our original analysis. So our results reported in the Result section are based on the comparisons between the best learning curves of different knowledge types as well as in the best model.

#### 2.4.1.2. An Alternative Variable for Accumulative Experience: Duration of Experience

We checked whether the results were robust to the operationalization of the experience variables. Our current experience variables represent the *counts* of prior experiences in the same domain (or same technology or same methodology) and do not consider the duration of experience. In software development, the period during which a member is assigned in a project is not same, and duration may affect the amount of knowledge gained from that experience. Therefore, simply counting the project numbers, the way we defined our variables, may not be sufficient. For example, experiencing a particular domain for 6 months (in a project) would be different from experiencing it for two years (in a project). To check whether this may be a concern, we developed an alternative measure for cumulative experience,

*ExperienceDuration*, which computes the cumulative duration, total days, for which an individual has worked on a particular knowledge.

$$ExperienceDuration = \left( \frac{\sum_{i}^{N} \sum_{j}^{P_i} k_{ij} \times (start\ date\ in\ project\ j - end\ date\ in\ project\ j)}{N \times M} \right)$$

We replaced the Experience variables with *ExperienceDuration* variables in Model 2, ran the regression, and found that the *ExperienceDuration* variables have the same as our original results in Hypotheses 1 and 3, but not in Hypothesis 2. Although we found differential learning curve estimates for domain, technology, and methodology (*DomainExperienceDuration*= -0.123, *TechnologyExperienceDuration*= -0.156, *MethodologyExperienceDuration*= -0.144), they were not significantly different (see Model 3 in Table 2-5).

**Table 2-5. Regression Results of Alternative Models**

| **Model 3** | | | **Model 4** | | |
|---|---|---|---|---|---|
| **Variables** | **Estimate** | **Std Err** | **Variables** | **Estimate** | **Std Err** |
| *Intercept* | 19.135*** | 0.276 | *Intercept* | 17.398*** | 0.4303 |
| *ProjectSize* | 0.152*** | 0.0186 | *Revenue* | 0.198*** | 0.0160 |
| *ln(OutsourcingRatio)* | 2.983*** | 0.1923 | *ln(OutsourcingRatio)* | 2.808*** | 0.1692 |
| *NumMethodologies* | 0.040*** | 0.0071 | *NumMethodologies* | 0.026*** | 0.0063 |
| *NumTechnologies* | 0.007*** | 0.0012 | *NumTechnologies* | 0.006*** | 0.0011 |
| *Y2005* | -0.403*** | 0.1073 | *Y2005* | -0.370*** | 0.0941 |
| *Y2006* | -0.261*** | 0.1006 | *Y2006* | -0.202** | 0.0890 |
| ln(*ExperienceDomainDuration*) | -0.123*** | 0.0316 | ln(*ExperienceDomain*) | -0.168*** | 0.0652 |
| ln(*ExperienceTechDuration*) | -0.156*** | 0.0508 | ln(*ExperienceTech*) | -0.487*** | 0.1087 |
| ln(*ExperienceMethodDuration*) | -0.144*** | 0.0434 | ln(*ExperienceMethod*) | -0.475*** | 0.0963 |
| *LapseDomain* | 0.999 | 0.0079 | ln(*LapseDomain*) | -0.064 | 0.0439 |
| *LapseTech* | 0.985 | 0.0234 | ln(*LapseTech*) | 0.082 | 0.0699 |
| *LapseMethod* | 0.941*** | 0.0250 | ln(*LapseMethod*) | 0.136** | 0.0536 |
| Sample Size (*N*) | 454 | | *n* | 454 | |
| $R^2$ | 0.7333 | | $R^2$ | 0.6729 | |

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

**Notes**: Robust standard errors are reported. $R^2$ for Model 3 is pseudo-$R^2$.

The reasons that we chose the counts of prior experiences as our original cumulative experience rather than the durations of experiences are that the count measures, the *Experience* variables, 1) produce a better model fitness ($R^2$ value: *Experience* = 0.7769 > *ExperienceDuration*= 0.7333) and 2) have been dominantly used in learning curve studies in software development (Boh et al. 2007; Langer et al. 2008; Huckman et al. 2009; Narayanan et al. 2009).

#### 2.4.1.3. An Alternative Forgetting Model: Power Function of Time

To test whether forgetting effects was robust to model specification, we tested an alternative forgetting model using the power function of time, which is another widely used function for fitting forgetting curves (Wixted and Ebbesen 1991).

$$y = cx^{\beta} t^{\alpha} \qquad \text{(eq. 3)}$$

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

The variables $y$, $x$, $c$ and $\beta$ are the same as in equations 1 and 2. Variable $t$ represents the time elapsed since last trial and $\alpha$ is the forgetting rate parameter. When forgetting curve effects exist, $\alpha$ takes a value between 0 and 1. Larger values of $\alpha$ implies a steeper slope of forgetting curves (i.e., faster forgetting).

The results were same as our original results. Forgetting curve effect exists only in methodology knowledge (see Model 4 in Table 2-5). This result provides further confidence that forgetting effects do exist in software development, especially for methodology knowledge.

The reasons that we chose the depreciation model as our original model rather than power function of time are 1) that the depreciation model appears better to represent the forgetting effects of each knowledge types by tightly coupling the lapse time and cumulative experience variables than the power

function model[7] 2) and we want to be consistent with the models of most organizational learning studies so that our results would be easily compared to any result of them.

#### 2.4.1.4. Heterogeneity of Project Team Members' Experience Levels

Our unit of analysis is project team. We operationalized the team-level experiences using the average of team members' experience levels. Although the average experience of project team members provides a reasonable proxy for project team, it also has a limitation that it cannot capture the distribution of team members' experience levels, which may also affect the performance of project team. For example, given an average level of team members' experiences, a project team can be composed in many ways, and performance of the team may vary by the way of composition.

For example, suppose there are two teams: One is a homogeneous team composed of team members who have the same (or similar) level of experience on the required knowledge and the other is a heterogeneous team composed of a small number of highly experienced members on the required knowledge and others who have relatively low level of experience. Given that two teams have the same level of team experience (i.e., the same average of members' experiences), which team would be better? We analyzed the effects of heterogeneity of project team members' experience levels and checked whether they affected our main findings if they exist.

We computed the standard deviation of team members' cumulative experiences on the same knowledge – domain (*DiversityDomain*), technology (*DiversityTechnology*), and methodology
For example, there are three members ($M_1$, $M_2$, and $M_3$) in a project team. If M1 has 10 cumulative experience on the same technology as defined above, M2 has 5, and M3 has nothing, the diversity among team members' experience (*DiversityTechnology*) becomes 5 (i.e., the standard deviation of 0, 5, and 10).

[7] The power function of time model allows that more than what has been accumulated (i.e., learned) can be depreciated (i.e., forgotten), while the depreciation model does not. Moreover, in the power function of time model, the experience variable ($x$) and lapse variable ($t$) for a particular knowledge type independently affect the dependent variable, while in the depreciation model, these two variables are interdependently affect the dependent variable. Because we use multiple knowledge types in single model

We added *Diversity* variables in Model 2 and ran the regression (see Model 5 in Table 2-6). Again, we observed that our original results regarding Hypotheses 1, 2, and 3 did not change by adding these *Diversity* variables.

With respect to the diversity variables for team members' experiences, we find that diversity of members' experiences with the same methodology and the same technology has a marginal negative effect on project performance (*DiversityMethodology*: $\gamma = 0.072$, $p < 0.01$; *DiversityTechnology*: $\gamma = 0.133$, $p < 0.01$). However, diversity in domain experience does not seem to degrade project performance (*DiversityDomain*: $\gamma = -0.010$).

Given the software development is a group task where a number of members work together, those members are supposed to possess a minimal level of knowledge. For this reason, having homogeneous team members would be more beneficial than having heterogeneous members because it would reduce the cost to be incurred for transferring knowledge among members.

Our next question naturally becomes whether the effects of diversity of team members' experience are different or nor for the types of software development knowledge – domain, technology, and methodology.

Regarding the differential diversity effects of software development knowledge types, the coefficient for *DiversityMethodology* was found to be significantly less than that of *ExperienceDomain* ($p < 0.1$). Similarly, the coefficient for *DiversityTechnology* was significantly less than that of *ExperienceDomain* ($p < 0.01$). However, the difference between *DiversityMethodology* and *DiversityTechnology* was not significant (ns)[8].

We use again the properties of procedural and declarative knowledge here. When heterogeneity among members' knowledge exist, knowledge transfer should occur on the spot. This transference usually occurs

[8] We used the 99%, 95%, and 90% confidence intervals of estimates for the comparisons.

via an intra-team meeting (or training) or one-to-one communication among team members, probably from experts to novices. Due to the time, space, and other resource constraints of on-the-spot practices, the transference usually forms an instruction, where declarative knowledge is more suitable to transfer than declarative knowledge, rather than an extensive hands-on exercise, where perhaps procedural knowledge is more transferred more efficiently.

**Table 2-6. Regression Results of the Heterogeneity of Experience Levels**

| **Model 5** | | |
|---|---|---|
| **Variables** | **Estimate** | **Std Err** |
| *Intercept* | 18.254*** | 0.1627 |
| *ProjectSize* | 0.153*** | 0.0165 |
| ln(*OutsourcingRatio*) | 2.814*** | 0.1721 |
| *NumMethodologies* | 0.023*** | 0.0065 |
| *NumTechnologies* | 0.006*** | 0.0011 |
| *Y2005* | -0.407*** | 0.0952 |
| *Y2006* | -0.227*** | 0.0893 |
| ln(*ExperienceDom*) | -0.238*** | 0.0812 |
| ln(*ExperienceTech*) | -0.631*** | 0.1230 |
| ln(*ExperienceMethod*) | -0.592*** | 0.1174 |
| *LapseDom* | 0.999 | 0.0037 |
| *LapseTech* | 1.005 | 0.0056 |
| *LapseMethod* | 0.987** | 0.0060 |
| *DiversityDomain* | -0.010 | 0.0213 |
| *DiversityTech* | 0.133*** | 0.0286 |
| *DiversityMethod* | 0.072*** | 0.0303 |
| Sample Size (*N*) | 454 | |
| $R^2$ | 0.7910 | |
| **Significance Levels**: * $p < 0.1$, ** $p < 0.05$, *** $p < 0.01$<br>**Notes**: Robust standard errors are reported. $R^2$ is pseudo-$R^2$. | | |

#### 2.4.1.5. Combined Experience

One of our main hypotheses is that software development knowledge is composed of multidimensional knowledge types – domain, technology, and methodology – and each type can be independently learned from experience with others (Hypothesis 1). In order to test this hypothesis, we defined the measures for cumulative experience separately for each knowledge type and found the separate learning effects of them. Statistical analysis such as multicollinearity and inter-correlation sufficiently showed that our measures and their effects are independent.

However, one may ask a question if there is any synergetic effect for combined experience which practices the same domain, methodology, and technology knowledge at single project. For example, suppose that a new project requires the online e-banking (domain), RUP (methodology), and J2EE (Technology) knowledge and there are two candidate members for the project. One has experienced the online e-banking, RUP, and J2EE at single project; the other has experienced them at three different projects such that online e-banking at project A, RUP at project B, and J2EE at project C. Based on our current measures, these two persons have the same level of experience. The questions would be whether they are really same, if they are different, whether our main results are affected.

In order to answer to the above questions, we are analyzing the data now.

## 2.5. Conclusion and Discussion

This study investigated the effects of software project team's prior experience on similar tasks in software development. This study examined the differential 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. In addition, this study examined the effects of diversity among team members' experience. Our results show that the difference among project team members' experiences on the methodology and technology required for developing software negatively affect the project team's performance while that on the domain does not affect performance.

### 2.5.1. Theoretical Implications

Our paper makes several contributions to literature. First, we included 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. Lastly, the analyses on the heterogeneity of experience levels among team members and the combined experience uniquely enriched our understanding on the effects of prior experience in software development.

### 2.5.2. Managerial Implications

Our empirical findings also shed light into practical questions in software development such as *how to staff software projects to maximize performance*, *how to develop and retain expertise,* and *how to think about career development paths for software professionals*.

Project staffing is a matching problem which matches the knowledge and skills of employees with the characteristics of software project. The normative staffing solution would be to staff the project team with only employees who have the required knowledge and skills relevant to the project at hand. However, in many organizations, this simple approach is not always feasible due to the limited human

resources and specialized employees' prior engagements on other projects. When employees who fully satisfy all multidimensional knowledge requirements of the project are not available, and among the available human resources, some have experience on domains, but not on technology or methodology, while some others may have experience on technology and methodology, but not on domain, organizations need to decide which employees should be assigned to the project. More generally, among domain, technology, and methodology knowledge, which would be most important in ensuring the success of a project? Our findings suggest that methodology and technology experiences precede the domain experience.

Another important question in project staffing is about the diversity of team. One basic question would be whether one highly expert or several moderately experienced workers are better? Our findings implies that several moderate experts is perhaps preferable than one highly experience guru, given the same average experience of whole team members at least in methodology and technology. However, regarding the domain knowledge, it does not matter.

Because of the learning and forgetting curve effects, project staffing is not just a matching problem to maximize the current performance of project but a knowledge development and retention problem. Organization and individuals accumulate knowledge and skills by conducting projects. On the other hand, as time goes, some cumulated knowledge and skills depreciate. Managers in software organization might be interested in the costs to develop experts and retain them in each of the knowledge types. From the perspective of learning by doing, domain knowledge is more difficult to be learned from experience than technology and methodology. An alternative interpretation is that it takes more time to be an expert in domain than in technology or methodology (if the organization relies solely on learning by doing). On the other hand, according to our findings on forgetting effects, retaining expertise in a particular methodology is more difficult than that on technology or domain because it is more subject to depreciation. In summary, to develop expertise in a particular domain is relatively difficult, but once it is

obtained, the expertise hardly decays, while to develop expertise in a particular methodology is relatively easy, but it is also easy to be depreciated.

### 2.5.3. Limitations and Future Research

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, functionality, 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 is that, in this paper, we examined only one type of experiences – direct experience on the same or similar tasks. There are different types of experiences such as the experience on different or unrelated tasks, relational experience (e.g., experience working together), and indirect experience. Last limitation is that our analysis is based on a single firm data which may contain unique characteristics of the firm and they may affect our results. In this regards, we encourage the replications of our analysis using other data set.

Obvious future research directions stem from the aforementioned limitations. One example is the examination of different types of experience. Prior experience may interfere with learning a new thing or experiencing on a new thing may facilitate forgetting the existing knowledge about different things. On the other hand, in some innovative and problem solving environment, different and diverse knowledge is found to enhance performance. Software development requires both aspects of the innovative problem which requires diverse knowledge and the routine problem which requires specialized skills and efficiency; therefore, it would be interesting to investigate whether breadth or diversity/variety of prior experiences have an impact on software development performance. As we did here, it would be interesting to investigate the differential effects of this diverse experience of software development knowledge types. 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?

Software development is a group task; therefore, the intrarelationship of project team should affect project team's performance. Intrarelationship can be affected by direct or indirect relational experience among members or social networks. In addition, we can define various types of relational experience based on the characteristics of relation or experience, and they may affect performance differently.

# CHAPTER 3. Specialist vs. Generalist: Who Is More Valuable in Software Development Projects?

## 3.1. Introduction

As software becomes ubiquitous and software technologies advance very fast, software development requires more professional knowledge, software practitioners are also required to become professionals, and these professionals, who possess the essential knowledge for software development, become a most critical success factor for software development. However, at the same time, due to the huge variety of software applications and rapidly developing software technologies, it is almost impossible for a software practitioner to become an omniscient professional, who knows everything about software development.

Researchers and practitioners seem to come to a consensus that professionals can be categorized into two types: a generalist, who has broad expertise on various subjects rather than deep expertise on a subject, and a specialist, who has deep expertise on a subject rather than broad expertise on various subjects. Each type of professional has its own pros and cons, and different areas may need different types of professional. In the context of software development projects, what types of professional, among specialist and generalist, is more desired? There has been an extensive debate on this question among software practitioners for a long time[9]. However, compared to this huge interest in practice, not much attention has been paid to this question by researchers. Therefore, we investigate this question in this study.

In this chapter, we conceptualize the types of professional using two different types of experience – specialized vs. diverse experience – in such a way that a generalist has diverse prior experience and a specialist has specialized prior experience. Very few studies have investigated the impacts of specialized

[9] See "Agile Developers: Generalists or Specialists?", Dr. Dobb's, 12 February 2002, http://www.cio.com/article/102352/Specialists_vs._Generalists; "Specialists vs. Generalists", CIO, 5 April 2007, http://www.ddj.com/architect/184415788.

and diverse experience together in the context of software development. To the best of our knowledge, only Boh et al. (2007) and Narayanan et al. (2009) studied both type of experience. While these initial studies has important implications for software development practices, we note some limitations that need to be further researched so as to provide more substantive managerial guidance to software practitioners.

First, although the previous studies have successfully found the effects of specialized and diverse experience in software development, they offer limited managerial implications as knowledge and experiences were modeled as uni-dimensional constructs (e.g., related vs. unrelated task). Software development is a complex knowledge work which requires multi-dimensional knowledge and skills. Given that software development requires a multi-dimensional set of knowledge and skills, and in practice, project staffing typically considers prior experiences in these dimensions, it is important to ascertain whether the specialized and diverse experience effects exist independently for different components of software development knowledge and skills. In this regard, we study the impacts of specialized and diverse experience at a finer-grained level of knowledge. We classify the software development knowledge into three types – domain, technology, and methodology – and investigate the impacts of specialized and diverse experience at this knowledge type level.

Second, software development is an interdependent group task. A group of people with different knowledge and skills, which we call a software project team, works together to develop software. A software project team consists of multiple roles and has a hierarchy among the roles; a project team typically consists of one or more project managers and their subordinate members such as designers, developers, and testers. Each role has a different set of responsibility and accountability, and requires a different set of knowledge and skills. Accordingly, each role may necessitate a different type of professional. However, previous studies did not consider these project roles; instead, they implicitly assumed all team members have an identical role and equally influence the team's performance. For this reason, we study the differential impacts of specialized and diverse experience by project role – project managers (PMs) vs. non-project managers (non-PMs).

Lastly, we believe that our research context, a custom software development environment, is better to test the impacts of diverse experience than that of previous studies, a software maintenance environment. Because of the nature of maintenance jobs, knowledge about the existing software system is essential to fix or upgrade that software system; therefore, software maintenances on a system are not conducted by a temporal project team but by a designated persistent team or group in general. 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 would repeat projects in the same domain, same technology and same methodology more frequently than in the new software development context. On the other hand, in software development, there is relatively greater variability in terms of knowledge and skills required across projects to which developers are assigned.

## 3.2. Theoretical Background and Empirical Questions

### 3.2.1. Specialized vs. Diverse Experience

It is known that a specialist, who has specialized experience but does not have diverse experience, can enhance performance. The effects of specialized experience (i.e., specialization) are supported by the learning curve literature (Dutton and Thomas, 1984; Yelle, 1979). Learning curves originally exist only when an identical task is repeatedly conducted. In software development projects, the task itself may be related and similar but is rarely the same; however, fortunately, prior literature successfully documented the learning curve effects in the context of software development (Langer et al. 2008; Huckman et al., 2009) and maintenance (Boh et al., 2007; Narayanan et al., 2009).

Specialization does not always provide a positive effect. It is well known that the marginal contribution of experience to performance gains of a learning curve decreases as the experience accumulates, and the learning curve becomes a plateau eventually (Argote and Epple, 1990). Additionally, it is also known that too much specialization sometimes leads to the learning myopia, which prohibits from learning new knowledge (Levinthal and March, 1993).

It is also known that a generalist, who has diverse experience but does not have specialized experience, can enhance performance. The diverse experience (i.e. diversity) is known to facilitate learning new knowledge (or conducting a new task). Absorptive capacity, the ability to assimilate new knowledge into existing knowledge asset, would be relevant evidence of the effects of diversity. Diverse prior knowledge is known to increase absorptive capacity by enhancing assimilative powers (Cohen and Levinthal, 1990). In addition, some learning curve literature found that the accumulated experience on related tasks, which can be considered as diverse experience in some sense, is better than those on the same task, and those on different tasks (Boh et al., 2007; Schilling et al., 2003). Narayanan et al. (2009) argued three reasons why diversity has positive effects on performance gains. According to their arguments, first, diversity can provide a contextual schema which helps us to assimilate a new thing. Second, diversity provides opportunities to learn how to learn something so that it enables us to reduce trials and errors when acquiring new things. Lastly, some variations on the jobs (e.g., job rotations) may provide us with more motivations, and may increase productivity eventually.

However, similar to specialization, it is also found that, in some cases, diversity may harm performance. Narayanan et al. (2009) found the inverted U shaped effects of diversity, which means that some diversity enhances performance, but too much diversity does not. Moreover, some literature in cognitive theory of knowledge transfer also argued that prior knowledge can hinder acquiring new knowledge when there is a conflict between old and new knowledge (Armstrong and Hardgrave, 2007).

### 3.2.2. Empirical Questions

Due to lack of consistent arguments in explaining when the positive effects of specialized or diverse experience or both exist and when the negative effects exist, we take an exploratory approach by investigating the following questions.

As it is aforementioned, software is a complex task, which requires multidimensional knowledge – domain, methodology, and technology. Are the most desired professional types for different types of

knowledge identical? We argue that, for some knowledge types, diversity may help to enhance the performance of software development, however, for some others, it may not help. Similarly, we argue that specialization may have the differential effects of knowledge types. Therefore, our first empirical question is whether the most desired professional type varies by knowledge type – domain, methodology, and technology.

A project team consists of multiple different project roles. Each project role requires a different set of expertise. We argue that some roles may need more specialized experience in some knowledge types, while some roles may need more diverse experience in them. Therefore, our second question is whether the most desired professional type varies by project role – PMs vs. non-PMs.

## 3.3. Methodology

In order to empirically answer to our questions, we used the same data set used in Chapter 2.

### 3.3.1. Measures

We measured software project performance, our dependent variable, using actual labor costs (*LaborCost*) as we did in Chapter 2 and use the same control variables for project size (*ProjectSize*), complexity (*NumTechnologies, NumMethodologies*), outsourcing (*OutsourcingRatio*), macroeconomic factors (*Y2005, Y2006*).

We computed measures for specialized experience (*SpecializedExperience*) and diverse experience(*DiverseExperience*). 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 (*SpecializedDomain*), same technology knowledge (*SpecializedTech*) or same methodology knowledge (*SpecializedMethod*). These measures are the same to the cumulative experience variables – *ExperienceDomain, ExperienceTech, ExperienceMethod* – used in Chapter 2. Taxonomy of domain, technology, and methodology was also explained in Chapter 2.

We also computed the unique number of different knowledge and skills, which he/she experienced in prior projects but are not required in the focal project, in domain, technology, and methodology, respectively, to measure the three diverse experience variables – *DiverseDomain, DiverseTech,* and *DiverseMethod*.

$$DiverseExperience = \left(\frac{\sum_{i=1}^{N} d_i}{N \times M}\right)$$

*N* is the number of employees in the focal project and *M* is the number of knowledge units required for the focal project. $P_i$ represents the number of prior projects in which member *i* participated, and $d_i$ is the unique number of knowledge units, which are not required for the focal project but project member *i* experienced in prior projects.

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 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 2 projects, which used $K_{M1}$ and $K_{M2}$, and 1 project, which used $K_{M2}$ and $K_{M3}$ ,and member $M_2$ was previously assigned to 1 project which only used $K_{M1}$, then the total number of projects experienced the methodologies either $K_{M2}$ or $K_{M3}$ by the project team is 6(=2*2+1+1) and the average number per team member is 3(=6/2). 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 1.5.

Similarly, the diverse experience at the team level is also normalized. Suppose that, besides $K_{M1}$ and $K_{M2}$, M1 has experienced 1 other methodology, say $K_{M3}$, and M2 has two, $K_{M3}$ and $K_{M4}$, in prior projects. Then M1's diverse experience in methodology becomes 1, M2's becomes 2, and so, the average of individual's diverse experience becomes 1.5. However, because there are 2 methodologies used in the project, the experience count is normalized by a factor of 2; therefore, the final measure for a team's diversity experience for methodology (*DiverseMethod*) becomes 0.75.

Specialized and diverse experiences of PMs and non-PMs are measured similarly to those of team. Only difference lies in calculating the average of individuals' experiences. Only PMs' experiences are averaged for the experiences of PMs[10], and only non-PMs' experiences are included in calculating the experiences of non-PMs. They are finally normalized by the number of knowledge and skills used in the focal project like the team's experience.

Tables 3-1 and 3-2 show the descriptive statistics and inter-correlations among variables: Table 3-1 includes the independent variables defined for team level, and Table 3-2 includes those for PMs and non-PMs.

[10] For some large projects, more than one PM can be assigned. In our data set, the average number of PMs is 1.07.

## Table 3-1. Descriptive Statistics and Inter-correlations (I) –Team Level

| | Mean | Stdev | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *1. LaborCost* | 19.32 | 1.59 | | | | | | | | | | | | |
| *2. ProjectSize* | 1.51 | 2.82 | 0.66*** | | | | | | | | | | | |
| *3. ln(Outsource)* | 0.42 | 0.22 | 0.47*** | 0.16*** | | | | | | | | | | |
| *4. NumMethod* | 12.21 | 6.88 | 0.51*** | 0.49*** | 0.07* | | | | | | | | | |
| *5. NumTech* | 43.97 | 46.8 | 0.60*** | 0.62*** | 0.02 | 0.50*** | | | | | | | | |
| *6. Y2005* | 0.38 | 0.49 | -0.03 | -0.1** | 0.05 | -0.07* | 0.09** | | | | | | | |
| *7. Y2006* | 0.34 | 0.48 | -0.01 | -0.01 | 0.04 | 0.02 | -0.01 | -0.56*** | | | | | | |
| 8. ln(*SpecDom*) | 1.09 | 0.84 | -0.35*** | -0.16*** | -0.05 | -0.17*** | -0.27*** | -0.18*** | 0.05 | | | | | |
| 9. ln(*SpecTech*) | 0.58 | 0.50 | -0.49*** | -0.30*** | 0.00 | -0.27*** | -0.43*** | -0.09** | 0.05 | 0.46*** | | | | |
| 10. ln(*SpecMethod*) | 0.63 | 0.58 | -0.44*** | -0.24*** | -0.05 | -0.16*** | -0.34*** | -0.12*** | 0.02 | 0.42*** | 0.61*** | | | |
| 11. ln(*DivDom*) | 1.18 | 0.54 | 0.13*** | 0.01 | 0.13*** | 0.12*** | 0.02 | 0.04 | 0.01 | -0.25*** | 0.00 | 0.06 | | |
| 12. ln(*DivTech*) | 0.62 | 0.63 | -0.46*** | -0.30*** | 0.04 | -0.26*** | -0.53*** | -0.13*** | -0.01 | 0.30*** | 0.52*** | 0.48*** | 0.11*** | |
| 13. ln(*DivMethod*) | 0.16 | 0.26 | -0.28*** | -0.18*** | 0.02 | -0.45*** | -0.23*** | -0.02 | -0.05 | 0.13*** | 0.23*** | 0.21*** | 0.05 | 0.34*** |

## Table 3-2. Descriptive Statistics and Inter-correlations (II) – PMs and Non-PM

| | Mean | Stdev | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *1. LaborCost* | 19.32 | 1.59 | | | | | | | | | | | | | | | | | | |
| *2. ProjectSize* | 1.51 | 2.82 | 0.66*** | | | | | | | | | | | | | | | | | |
| *3. ln(OutsourcingRatio)* | 0.42 | 0.22 | 0.47*** | 0.16*** | | | | | | | | | | | | | | | | |
| *4. NumMethodologies* | 12.21 | 6.88 | 0.51*** | 0.49*** | 0.07* | | | | | | | | | | | | | | | |
| *5. NumTechnologies* | 43.97 | 46.8 | 0.60*** | 0.62*** | 0.02 | 0.50*** | | | | | | | | | | | | | | |
| *6. Y2005* | 0.38 | 0.49 | -0.03 | -0.10** | 0.05 | -0.07* | 0.09** | | | | | | | | | | | | | |
| *7. Y2006* | 0.34 | 0.48 | -0.01 | -0.01 | 0.04 | 0.02 | -0.01 | -0.56*** | | | | | | | | | | | | |
| 8. ln(*PMSpecDom*) | 0.48 | 0.31 | -0.19*** | -0.07 | -0.06 | -0.08* | -0.13*** | -0.21*** | 0.11** | | | | | | | | | | | |
| 9. ln(*PMSpecTech*) | 0.57 | 0.62 | -0.41*** | -0.24*** | 0.01 | -0.23*** | -0.37*** | -0.05 | 0.03 | 0.24*** | | | | | | | | | | |
| 10. ln(*PMSpecMeth*) | 0.87 | 0.83 | -0.30*** | -0.16*** | -0.08* | 0.00 | -0.23*** | -0.14*** | 0.00 | 0.22*** | 0.44*** | | | | | | | | | |
| 11. ln(*NonPMSpecDom*) | 1.15 | 0.92 | -0.32*** | -0.16*** | -0.03 | -0.14*** | -0.27*** | -0.20*** | 0.06 | 0.57*** | 0.31*** | 0.28*** | | | | | | | | |
| 12. ln(*NonPMSpecTech*) | 0.46 | 0.39 | -0.32*** | -0.28*** | 0.04 | -0.23*** | -0.37*** | -0.10** | 0.07 | 0.14*** | 0.24*** | 0.20*** | 0.32*** | | | | | | | |
| 13. ln(*NonPMSpecMeth*) | 0.29 | 0.33 | -0.02 | -0.04 | 0.01 | -0.10** | -0.08* | -0.08* | 0.00 | 0.02 | -0.09** | -0.01 | 0.17*** | 0.35*** | | | | | | |
| 14. ln(*PMDivDom*) | 1.25 | 0.75 | 0.15*** | 0.02 | 0.12*** | 0.08* | 0.04 | 0.02 | -0.03 | -0.11*** | 0.05 | 0.11*** | -0.14*** | 0.04 | 0.05 | | | | | |
| 15. ln(*PMDivTech*) | 0.60 | 0.65 | -0.44*** | -0.29*** | 0.06 | -0.27*** | -0.50*** | -0.12*** | 0.02 | 0.13*** | 0.48*** | 0.37*** | 0.25*** | 0.30*** | 0.02 | 0.12*** | | | | |
| 16. ln(*PMDivMeth*) | 0.16 | 0.27 | -0.25*** | -0.15*** | -0.04 | -0.33*** | -0.19*** | -0.10** | -0.03 | 0.05 | 0.21*** | 0.25*** | 0.05 | 0.08* | 0.06 | 0.11** | 0.36*** | | | |
| 17. ln(*NonPMDivDom*) | 1.11 | 0.57 | 0.12*** | 0.03 | 0.15*** | 0.11** | 0.03 | 0.01 | 0.06 | -0.17*** | 0.05 | 0.07 | -0.17*** | 0.14*** | 0.12*** | 0.41*** | 0.14*** | 0.09* | | |
| 18. ln(*NonPMDivTech*) | 0.54 | 0.58 | -0.41*** | -0.28*** | 0.02 | -0.23*** | -0.50*** | -0.17*** | -0.05 | 0.14*** | 0.32*** | 0.27*** | 0.31*** | 0.41*** | 0.21*** | 0.09* | 0.76*** | 0.25*** | 0.19*** | |
| 19. ln(*NonPMDivMeth*) | 0.14 | 0.23 | -0.20*** | -0.17*** | 0.09* | -0.44*** | -0.21*** | 0.07 | -0.07 | -0.02 | 0.12** | -0.1** | 0.07 | 0.20*** | 0.31*** | 0.05 | 0.16*** | 0.22*** | 0.07 | 0.33*** |

### 3.3.2. Analytical Approach

The most common way to investigate the effects of accumulated experience (i.e., specialized experience) is to use learning curves. Learning curves have been successfully used in the context of software development (Huckman et al., 2009; Langer et al., 2008) and maintenance (Boh et al., 2007; Narayanan et al., 2009). In addition, the effects of diverse experience were examined successfully along with the learning curves (Narayanan et al., 2009). We develop the model to be consistent with the prior literature on learning curve effects (Boh et al. 2007; Huckman et al. 2009; Langer et al. 2008) as well as on diverse experience effects (Narayanan et al., 2009) in software development.

**Model 1**:

$$\ln(LaborCost_i) = \beta_0 + \beta_1 \times ProjectSize_i + \beta_2 \times \ln(OutsourcingRatio_i) + \beta_3 \times NumMethodologies_i + \beta_4 \times NumTechnologies_i + \beta_5 \times Y2005_i + \beta_6 \times Y2006_i + \beta_7 \times \ln(SpecializedDomain_i) + \beta_8 \times \ln(SpecializedTech_i) + \beta_9 \times \ln(SpecializedMethod_i) + \beta_{10} \times \ln(DiverseDomain_i) + \beta_{11} \times \ln(DiverseTech_i) + \beta_{12} \times \ln(DiverseMethod_i) + \varepsilon_i$$

Actually, the diverse experience is not cumulated experience; however, prior literature of learning curves usually uses log transformations for any experience related variables as well as control variables. For example, Faraj and Sproull (2000) incorporated the shared experience, which is not cumulative experience but just a ratio, into a learning curve formula using a log transformation. In order to investigate the effects of specialized and diverse experience by project role, we define Model 2.

**Model 2**:

$$\ln(LaborCost_i) = \beta_0 + \beta_1 \times ProjectSize_i + \beta_2 \times \ln(OutsourcingRatio_i) + \beta_3 \times NumMethodologies_i + \beta_4 \times NumTechnologies_i + \beta_5 \times Y2005_i + \beta_6 \times Y2006_i + \beta_6 \times \ln(PMSpecializedDomain_i) + \beta_7 \times \ln(PMSpecializedTech_i) + \beta_8 \times \ln(PMSpecializedMethod_i) + \beta_9 \times \ln(PMDiverseDomain_i) + \beta_{10} \times \ln(PMDiverseTech_i) + \beta_{11} \times \ln(PMDiverseMethod_i) + \beta_{12} \times \ln(NonPMSpecializedDomain_i) +$$

$\beta_{13} \times \ln(NonPMSpecializedTech_i) + \beta_{14} \times \ln(NonPMSpecializedMethod_i) +$

$\beta_{15} \times \ln(NonPMDiverseDomain_i) + \beta_{16} \times \ln(NonPMDiverseTech_i) +$

$\beta_{17} \times \ln(NonPMDiverseMethod_i) + \varepsilon_i$

### 3.3.3. 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) and conditional indice (CI) are within recommended limits.

## 3.4. Analysis and Results

Empirical results are summarized in Table 3-3. We conducted the analyses using a hierarchical approach. We first estimate a baseline regression model that only includes the control variables (Base Model) and progressively add the independent variables in subsequent models. In Model 1, we add the independent variables for the team level's specialized and diverse experience, and, in Model 2, we add the specialized and diverse experience variables for PMs and non-PMs. First, we check to see if the inclusion of the independent variables increases the explanatory power of the regression models. The increased explanatory power between models shows that specialized and diverse experience variables are significant predictors of the project performance gains, labor cost (Base model vs. Model 1: $\Delta R^2 = 0.104$, $\chi^2 = 206.85$, $p < 0.01$; base model vs. Model 2: $\Delta R^2 = 0.108$, $\chi^2 = 130.90$, $p < 0.01$). The base model has already explained in the previous chapter.

**Table 3-3. Regression Results**

| | Baseline | | Model 1 | | Model 2 | |
|---|---|---|---|---|---|---|
| **Variables** | **Estimate** | **Std. Err.** | **Estimate** | **Std. Err.** | **Estimate** | **Std. Err.** |
| *Intercept* | 17.109*** | 0.1268 | 18.228*** | 0.1674 | 18.302*** | 0.1722 |
| *ProjectSize* | 0.178*** | 0.0354 | 0.171*** | 0.0274 | 0.162*** | 0.0155 |
| *ln(OutsourcingRatio)* | 2.891*** | 0.2096 | 2.817*** | 0.1764 | 3.013*** | 0.1726 |
| *NumMethodologies* | 0.036*** | 0.0068 | 0.026*** | 0.0061 | 0.022*** | 0.0071 |
| *NumTechnologies* | 0.011*** | 0.0017 | 0.005*** | 0.0011 | 0.005*** | 0.0011 |
| *Y2005* | -0.221** | 0.1010 | -0.474*** | 0.0867 | -0.427*** | 0.0932 |
| *Y2006* | -0.187* | 0.0987 | -0.297*** | 0.0851 | -0.376*** | 0.0904 |
| ln(*SpecializedDomain*) | | | -0.152*** | 0.0503 | | |
| ln(*SpecializedTech*) | | | -0.357*** | 0.0950 | | |
| ln(*SpecializedMethod*) | | | -0.329*** | 0.0798 | | |
| ln(*DiverseDomain*) | | | 0.244*** | 0.1068 | | |
| ln(*DiverseTech*) | | | -0.401*** | 0.1138 | | |
| ln(*DiverseMethod*) | | | -0.164 | 0.2731 | | |
| ln(*PMSpecializedDomain*) | | | | | 0.149 | 0.1327 |
| ln(*PMSpecializedeTech*) | | | | | -0.170** | 0.0747 |
| ln(*PMSpecializedMethod*) | | | | | -0.083* | 0.0509 |
| ln(*NonPMSpecializedDomain*) | | | | | -0.250*** | 0.0498 |
| ln(*NonPMSpecializedTech*) | | | | | -0.109 | 0.1069 |
| ln(*NonPMSpecializedMethod*) | | | | | 0.087 | 0.1155 |
| ln(*PMDiverseDomain*) | | | | | 0.119** | 0.0529 |
| ln(*PMDiverseTech*) | | | | | -0.212** | 0.1051 |
| ln(*PMDiverseMethod*) | | | | | -0.059 | 0.1670 |
| ln(*NonPMDiverseDomain*) | | | | | 0.047 | 0.0711 |
| ln(*NonPMDiverseTech*) | | | | | -0.302*** | 0.1076 |
| ln(*NonPMDiverseMethod*) | | | | | -0.301 | 0.2207 |
| *n* | 556 | | 551 | | 460 | |
| $R^2$ | 0.667 | | 0.7708 | | 0.775 | |
| *Adjusted* $R^2$ | 0.664 | | 0.7657 | | 0.766 | |
| *CI* | 2.726 | | 3.635 | | 5.028 | |

### 3.4.1. Knowledge Types

With Model 1, we first observe that the coefficients of all control variables remain relatively stable – an indicator that multicollinearity is not at play. As we found in previous chapter, we find that all three specialized experience variables have significant effects on performance gains (*SpecializedDomain*: $\beta$ = -0.151, $p < 0.1$; *SpecializedTech*: $\beta$ = -0.357, $p < 0.01$; and *SpecializedMethod*: $\beta$ = -0.329, $p < 0.01$). With respect to diverse experience, diverse experience variables for technology and methodology have significant effects on performance gains (*DiverseTech*: $\beta$ = -0.401, $p < 0.01$; and *DiverseMethod*: $\beta$ = -0.164, $p < 0.01$), but diverse experience on domain deteriorates the project performance (*DiverseDomain*: $\beta$ = 0.244, $p < 0.01$). The results imply that the most desired professional for software development is a T-shaped professional (i.e., a mix of specialist and generalist) on technology and methodology (who has specialized experience on the required technology and methodology for the focal project as well as diverse experience on different technologies and methodologies) (Hansen and Oetinger, 2001) and a specialist on domain (who has specialized experience on the required domain for the focal project) (see Table 3-4).

**Table 3-4. Desired Professional Type: Knowledge Type**

| **Knowledge Type** | **Experience** | | **Desired Professional Type** |
|---|---|---|---|
| | **Specialized** | **Diverse** | |
| Domain | + | - | Specialist but **not** generalist |
| Technology | + | + | Specialist and generalist (= T-shaped) |
| Methodology | + | • | Specialist |
| + : increases performance (= decreases labor cost)<br>- : decreases performance (= increases labor cost)<br>• : does not affect performance | | | |

### 3.4.2. Project Roles

With Model 2, we observe again that the coefficients of all control variables, as well as specialized and diverse experience variables remain relatively stable. With respect to PMs' specialized experience, we find that specialized experience variables for technology and methodology have significant effects on project performance gains (*PMSpecializedTech*: $\beta$ = -0.107, $p < 0.01$; and *PMSpecializedMethod*: $\beta$ = -0.083, $p < 0.1$). However, specialized experience on domain does not affect the project performance (*PMSpecializedDomain*: $\beta$ = 0.149, $p < 0.1$). Regarding the PM's diverse experience, again, diverse experience on technology enhances the project performance (*PMDiverseTech*: $\beta$ = -0.211, $p < 0.05$); diverse experience on methodology does not affect the performance (*PMDiverseMethod*: $\beta$ = -0.059, *ns*); but diverse experience on domain negatively affects the project performance (*PMDiverseDomain*: $\beta$ = 0.119, $p < 0.05$). Overall results imply that the most desired professional type for PMs is a T-shaped professional on technology (who has specialized experience on the required technology for the focal project as well as diverse experience on different technologies), and a specialist on methodology required for the project. However, a person who has diverse experience on different domains would not be appropriate for the PMs.

When it comes to non-PMs' experience, specialized experience on domain and technology positively affects the project performance (*NonPMSpecializedDomain*: $\beta$ = -0.250, $p < 0.01$) and), while specialized experience on technology and methodology does not (*NonPMSpecializedTech*: $\beta$ = -0.109, *ns*; *NonPMSpecializedMethod*: $\beta$ = 0.087, *ns*). Regarding non-PMs' diverse experience, diverse experience on technology enhances the project performance (*NonPMDiverseTech*: $\beta$ = -0.302, $p < 0.01$), but that on domain and methodology was found to be an insignificant predictor (*NonPMDiverseDomain*: $\beta$ = 0.047, *ns*; *NonPMDiverseMethod*: $\beta$ = -0.301, *ns*). As a result, the most desired professional type for non-PMs is a generalist on technology (who has diverse experience on different technologies) and a specialist on domain (who has specialized experience on the required domain knowledge for the focal project) (see Table 3-5).

**Table 3-5. Desired Professional Type: Knowledge Type and Project Role**

| | PMs | | | Non-PMs | | |
|---|---|---|---|---|---|---|
| **Knowledge Type** | **Experience** | | **Desired Professional Type** | **Experience** | | **Desired Professional Type** |
| | **Specialized** | **Diverse** | | **Specialized** | **Diverse** | |
| Domain | ● | - | **Not** generalist | + | ● | Specialist |
| Technology | + | + | Specialist and generalist (= T-shaped) | ● | + | Generalist |
| Methodology | + | ● | Specialist | ● | ● | Any type |

+ : increases performance (= decreases labor cost)

- : decreases performance (= increases labor cost)

● : does not affect performance

### 3.4.3. 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 specialized and diverse experience variables – were found to be stable and were not affected by adding or dropping control variables. In order to further check the appropriateness of model specifications and variables definitions, we tested some alternatives for the variables.

## 3.5. Conclusion and Discussion

This paper raises a practical question, what types of professional are needed in software development projects. In order to answer this question, this paper empirically investigates the impacts of two types of experience – specialized and diverse experience – at a project team level. Our study has a set of unique contributions compared to prior literature, in sense that it considers the multi-dimensional knowledge domain, technology, and methodology knowledge) and the project roles (i.e., PMs vs. non-PMs) in software development.

This paper also raises some important open questions which draw the attentions of researchers to developing new theories. Software development is one representative example of a complex group knowledge task where a variety of knowledge is involved and professional knowledge workers co-work according to the defined their roles. Other examples of this type of group task include new product development, construction engineering, consulting, surgery and medical treatment, and so on. Our results imply that, in such a group knowledge task, the impacts of group member's specialized and diverse experience on a workgroup performance seem to be different by the nature of knowledge required and the individual's role within the group.

As being consistent with the previous learning curve literature in software development, our empirical results show that specialized experience enhances the software project performance in most cases except PMs' specialized experience on domain and non-PMs' specialized experience on methodology and technology, which may not affect the project performance.

It is obvious that specialized methodology knowledge is essential to PMs, who manage the overall process of software development. However, non-PMs should possess, at least, a part of methodology knowledge, which is required to perform their tasks. The field test company uses well-documented standard methodologies and emphasizes on training employees for its methodologies. We argue that such practices of the company possibly fulfill the necessitated basic methodology knowledge for non-PMs to perform their individual tasks which decrease the marginal contribution of learning by doing, while PMs still need to acquire implicit knowledge on how to coordinate and manage project activities and resources, which is hardly acquired by written documents or trainings.

Deep expertise on technology required for the project seems to be more necessitated to PMs, who make major decisions on the overall technical/software architecture and the development standards of the system than non-PMs who make and test codes according to the defined architecture and standards, which is relatively simple and routine job. Surely, non-PMs should have a certain level of knowledge to conduct

assigned jobs successfully; however, similar to methodology, technology is usually well documented in internal or external sources like books and online community, and relatively many training courses are provided for technology knowledge. Such documentation and training may provide the minimal required level of knowledge for non-PMs to conduct their jobs.

With respect to diverse experience, only technology experience appears to enhance the project performance; methodology was insignificant; and domain has negative effects. Especially, PMs' diverse experience on domain was found to negatively affect the project performance, although non-PMs' diverse experience does not seem to affect the project performance significantly.

There is a potential explanation for the negative effects of PM's diverse experience on domain. Some PMs, who have too much experience on different domains of software, may overconfidently believe that they know exactly what customers want, and may override the system features determined by non-PMs, who actually interact closely with customers, and customers. PMs usually do not gather the system requirements from customers; however, PMs may be involved in the final decisions on system features. Sometimes, PMs may add some unnecessary features or modify the customer's requirements at their disposal, which may worsen the overall project performance.

Prior related knowledge or experience may help people to acquire new knowledge providing a schematic context (Boh et al., 2007; Narayanan et al., 2009; Schilling et al., 2003); however, when the prior knowledge or experience conflicts with new knowledge, prior knowledge or experience prevents people from acquiring new knowledge (Amstrong and Hardgrave, 2007). Technologies, although they deal with more various problem contexts, many related technologies share the same problem context (e.g., various database management systems for database, various operating systems for computers, etc.). Perhaps, due to this shared context among methodologies (and some related technologies), diverse experience could facilitate the acquisition of new knowledge. On the other hand, domains, at least by our definition, are relatively exclusive of each other compared to technologies or methodologies; therefore, experience on

other domains does not seem to provide a contextual schema knowledge which facilitates acquiring new domain knowledge, and sometimes this predominant knowledge on different domains even may deter gathering right requirements (negative effect).

# CHAPTER 4. Software development project selection: the stochastic multiple-secretary problem with deferred decisions

## 4.1. Introduction

In this paper, we solve the software development project selection problem of an IT service company. IT service companies provide the software development outsourcing services to customers. Usually, an outsourcing contract is made by a competitive bidding process. However, sometimes, customers, who have the needs for the software development, directly contact a specific IT service company and request the outsourcing service; and, the company decides to accept or not this request. From the perspective of the IT service company, due to the limited resources, it cannot accept all requests from customers, but select some requests. Customers' outsourcing requests (i.e., software project leads) are not known to the IT service company a priori, but assumed to follow probability models. The IT service company does not have to decide to agree or not to conduct the project upon its request from the customer, but it does not know how long the customer will wait for the decision, either. The customer may find other company who is willing to provide the service in the meanwhile. We model this project selection problem as the *stochastic multiple-secretary problem with deferred decisions*.

The *secretary problem* is one of most studied problems in operations research (see Freeman, 1983 and Ferguson, 1989). In the secretary problem, items (i.e., candidates of secretary; in our context, IS development project requests) arrive over time, and the objective is to maximize the probability of choosing the best item or to maximize the expected value of the chosen item. The *multiple-secretary problem* (sometimes referred as the *k*-secretary problem) is to choose multiple items (i.e., *k* best items) sequentially. In the traditional secretary problem, the total number of items is usually given a priori; however, in some cases, this number is not given; instead, the item types and probability model for the arrivals of each item type are assumed to be known a priori. We call this problem the *stochastic secretary problem* (sometimes, referred as the *secretary problem with an unknown number of items*). The objective

of the stochastic secretary problem is to select the best items given a time deadline (i.e., during a given decision horizon).

One common assumption among various secretary problems is the instantaneous decision. It means the acceptance decision on an item needs to be made immediately upon the arrival of the item. However, we find that this assumption may be too strict in some practical problems (see the examples below). Therefore, we study a new problem called the *stochastic multiple-secretary problem with deferred decisions*. In this problem, we assume that items stay in the system and the sojourn time of an item – the amount of time an item is expected to spend in the system – is randomly drawn from a probability distribution. Unlike the traditional secretary problem, in our problem, the decision on an item can be deferred as long as the item stays in the system. We call this type of decision the *deferred decision*.

## 4.2. Related Problems

The online Knapsack problem is similar to our problem. Like the secretary problem, in the online Knapsack problem, items arrive over times, and the objective is to select most valuable items to fill the Knapsack, which has the limited capacity (Lueker, 1998; Marchettispaccamela & Vercellis, 1995). Secretary problem is a special case of the online Knapsack problem. In the secretary problem, items have equal sizes; however, in the online Knapsack, items may have different sizes. Most online Knapsack problems also assume that the total number of items is given. However, some studies assume that there is a decision horizon (i.e., time deadline) and items arrive according to the Poisson distribution during the decision horizon, instead of assuming that the number of items which will arrive is given (Kleywegt & Papastavrou, 1998; Papastavrou, Rajagopalan, & Kleywegt, 1996; Ross & Tsang, 1989; van Slyke & Young, 2000). Some literature provides the closed form of optimal policy (Kleywegt & Papastavrou, 1998); however, due to the complexity of the problem, it is not easy to get the optimal policy for this significant problem. One model (Ross & Tsang, 1989) assumes not only the arrival time of an item, but also the sojourn time of the item affect the decision of whether to accept the item or not and the sojourn

times of items are also random variables drawn from the Exponential distribution. Our problem has similar assumptions to Ross and Tsang (1989); however, our model is different from their model, because Ross and Tsang (1989) assume that the decision, whether to accept the arrived item, should be decided upon the arrival of the item like other literature. To the best of our knowledge, our model is unique and has not been addressed by any literature.

### 4.3. Model Description and General Results

Suppose there is an IT service company. The company receives the requests for software development project, which we call *items* hereafter, and decides whether to accept them. We define the software development project selection problem as $P_l(n; t, \boldsymbol{K_l})$. $l$ is the number of item types ($l > 0$), and $n$ is the number of items ($n \geq 0$) which can be accepted at most during the remaining time $t$. Item arrive at the system, stay, and leave the system randomly. $\boldsymbol{K_l}$ is a $l$-elements vector, $\boldsymbol{K_l} = \{k_1, k_2, \dots, k_l\}$, where $k_a$, $a$= 1, 2, …, $l$, is a non-negative integer and represents the number of $a$-type items currently staying in the system. Let $V_a$, $a$= 1, 2, …, $l$, be the value earned by accepting an $a$-type project. Without loss of generality, we assume that $V_a < V_{a+1}$ for $a$= 1, 2, …, $l$-1. The objective of $P_l(n; t, \boldsymbol{K_l})$ is to maximize the expected value by selectively accepting items during the remaining time $t$. An item can be accepted as long as it stays in the system; however, items which have already left the system cannot be accepted.

Let $E_l(n; t, k_l, \boldsymbol{K_{l-1}})$ be the maximum expected value which can be obtained by not accepting any items except $l$-type items at remaining time t (i.e., when time $t$ is left), with $n$, $k_l$, and $\boldsymbol{K_{l-1}}$. Any $l$-type items in the system at remaining time $t$ ($k_l > 0$) must be accepted, however, at most, up to $n$ number of items. Therefore, if $k_l \geq n$, $E_l(n; t, k_l, \boldsymbol{K_{l-1}}) = nV_l$. Otherwise (i.e., $k_l < n$), we can define the following recursive equation:

$$E_l(n; t, k_l, \boldsymbol{K_{l-1}}) = k_l V_l$$
$$+\sum_{i=0}^{n-k_l-1}\left\{\left(iV_l + \sum_{j=0}^{k_{l-1}} E_{l-1}(n-k_l-i; t, k_{l-1}-j, K_{l-2})\, P_D(j, k_{l-1}, l-1, t)\right) P_A(i, l, t)\right\}$$
$$+(n-k_l)V_l\left(1-\sum_{i=0}^{n-k_l-1} P_A(i, l, t)\right) \qquad \text{for } l = 2, 3, \ldots, \infty \qquad (1)$$

$k_l V_l$ is the value obtained by immediately accepting $k_l$ number of $l$-type items in the system at remaining time $t$. $\left(iV_l + \sum_{j=0}^{k_{l-1}} E_{l-1}(n-k_l-i; t, k_{l-1}-j, \boldsymbol{K_{l-2}})\, P_D(j, k_{l-1}, l-1, t)\right) P_A(i, l, t)$ is the maximum expected value when $i$ number of $l$-type items arrive during the remaining time $t$. Given the number of arrivals of $l$-type items (i.e., $i$), the maximum expected value can be calculated by solving the sub problems with ($l$-1) item types. Because $E_l(n; t, k_l, \boldsymbol{K_{l-1}})$ assumes to reject any items except $l$-type items, some other type items, which are in the system at remaining time $t$, may become unavailable (i.e., leave the system) very soon (i.e., during $dt$). $E_{l-1}(n-k_l-i; t, k_{l-1}-j, \boldsymbol{K_{l-2}}) P_D(j, k_{l-1}, l-1, t)$ represents the maximum expected value of the sub problem with ($l$-1) item types, when $j$ number of ($l$-1)-type items have left the system.

$P_A(i, a, t)$, $i = 0, 1, \ldots, n$ and $a = 1, 2, \ldots, l$, is the probability that $i$ number of $a$-type items arrive during the remaining time $t$. $a$-type items arrive randomly drawn from the Poisson distribution with the arrival rate $\lambda_a$; therefore, $P_A(i, a, t) = \frac{(\lambda_a t)^i e^{-\lambda_a t}}{i!}$. $P_D(j, k_a, a, t)$, $j$=0, 1, …, $k_a$, is the probability that $j$ among $k_a$ number of $a$-type items leave the system during [$t$, $t$+$dt$]. Therefore, $P_D(j, k_a, a, t) = \mathrm{C}_{\mathrm{j}}^{k_a} h_a(t)dt^j(1 - h_a(t)dt)^{k_a-j} = \frac{k_a!}{j!(k_a-j)!}(\mu_a dt)^j(1-\mu_a dt)^{k_a-j}$. $h_a(t)$ is the hazard rate of $a$-type item at remaining time $t$. We assume $a$-type items depart the system randomly drawn from the Poisson distribution with the departure rate $\mu_a$. In other words, the sojourn time of an $a$-type item is randomly drawn from the Exponential distribution with the mean sojourn time $\frac{1}{\mu_a}$. This assumption leads us to have a time-independent hazard rate, $h_a(t) = \mu_a$ for $a$-type item, $a$=1,2,…, $l$. Furthermore, we can define a constant,

$d_a = \mu_a dt$, such that $0 \le d_a \le 1$, which represents the probability that an $a$-type item departs during [$t$, $t+dt$]. Then, $P_D(j, k_a, a, t) = \frac{k_a!}{j!(k_a-j)!}(d_a)^j(1-d_a)^{k_a-j}$. Now we can rewrite Equation (1) as follows:

$$
\begin{aligned}
E_l(n; t, k_l, \boldsymbol{K_{l-1}}) &= nV_l \\
&- \sum_{i=0}^{n-k_l-1} \Bigg( (n-k_l-i)V_l \\
&- \sum_{j=0}^{k_{l-1}} E_{l-1}(n-k_l-i; t, k_{l-1}-j, \boldsymbol{K_{l-2}}) \frac{k_{l-1}!}{j!\,(k_{l-1}-j)!}(d_{l-1})^j(1 \\
&- d_{l-1})^{k_{l-1}-j} \Bigg) \frac{(\lambda_l t)^i e^{-\lambda_l t}}{i!}
\end{aligned}
$$

for $l$= 2, 3, …, ∞ (2)

There are boundary conditions of $E_l(n; t, k_l, K_{l-1})$.

B-1) $E_l(0; t, k_l, \boldsymbol{K_{l-1}}) = 0, \quad \forall t \; and \; \forall l$

B-2) $E_1(n; t, k_1, \boldsymbol{K_0}) = \min(\lambda_1 t + k_1, n)V_1$

Using these conditions and the recursive equation, Equation (2), we can calculate $E_l(n; t, k_l, \boldsymbol{K_{l-1}})$. The computational complexity of calculating $E_l(n; t, k_l, \boldsymbol{K_{l-1}})$ is $O((n-1)^{2(l-1)})$.

Properties of $E_l(n; t, k_l, \boldsymbol{K_{l-1}})$:

P1-1) $E_l(n; t, k_l, \boldsymbol{K_{l-1}})$is non-decreasing in $n$.

P1-2) $E_l(n; t, k_l, \boldsymbol{K_{l-1}})$is non-decreasing in $t$.

P1-3) $E_l(n; t, k_l, \boldsymbol{K_{l-1}})$is non-decreasing in $k_a$, $a$=1,2,…, $l$.

P1-4) $E_l(n; t, k_l, \boldsymbol{K_{l-1}})$is non-increasing in $d$.

Proof: All properties are intuitive.

*A basic decision rule*: At a given time $t$, if $E_l(n; t, k_l, \boldsymbol{K_{l-1}}) \leq \max_{a\in\{1,2,\ldots,l-1\}}[E_l(n-1; t, k_l, \boldsymbol{K_{l-1}} - \boldsymbol{e_a}) + V_a]$, where $\boldsymbol{e_a}$ is a unit vector whose $a^{th}$ element is 1 and the other elements are 0s, it is better to accept one item of which type is $a^* = \text{argmax}_{a\in\{1,2,\ldots,l-1\}}[E_l(n-1; t, k_l, \boldsymbol{K_{l-1}} - \boldsymbol{e_a}) + V_a]$ and which stays in the system than waiting for the arrival of new $l$-type item and not accepting any other items. Please note that $a^*$ may not be necessarily the ($l$-1)-type.

```
Calculate t_l^*(n, k_l, K_{l-1}) and a_l^*(n, k_l, K_{l-1});
Set t^* = t_l^*(n, k_l, K_{l-1});
Set a^* = a_l^*(n, k_l, K_{l-1});
While (t > 0 and n > 0) {  // t is the current remaining time.
        If (t ≤ t_l^*) {
                Accept one a^*-type item among k_{a^*};
                Update (n, k_l, K_{l-1});
                Calculate t_l^*(n, k_l, K_{l-1}) and a_l^*(n, k_l, K_{l-1});
                Set t^* = t_l^*(n, k_l, K_{l-1});
                Set a^* = a_l^*(n, k_l, K_{l-1});
        }
        Else if (t > t_l^*) {
                If an item arrives or departs {
                        If the item arrived is l-type, accept the item;
                        Update (n, k_l, K_{l-1});
                        Calculate t_l^*(n, k_l, K_{l-1}) and a_l^*(n, k_l, K_{l-1});
                        Set t^* = t_l^*(n, k_l, K_{l-1});
                        Set a^* = a_l^*(n, k_l, K_{l-1});
                }
                Else, wait (do nothing).
        }
}
```

**Figure 4-1. Optimal Decision Rule for the Problem with *l*, *n*, *t*, $k_l$, and $K_{l-1}$**

Let $t_l^*(n, k_l, \boldsymbol{K_{l-1}}; a)$ be the solutions of the following equations, $E_l(n; t, k_l, \boldsymbol{K_{l-1}}) = E_l(n-1; t, k_l, \boldsymbol{K_{l-1}} - \boldsymbol{e_a}) + V_a$ for $a$=1, 2, …, $l$-1. Then, we can define,

$t_l^*(n, k_l, \boldsymbol{K_{l-1}}) = \max_{a\in\{1,2,\ldots,l-1\}}[t^*(n, k_l, \boldsymbol{K_{l-1}}; a)]$ and

$a_l^*(n, k_l, \boldsymbol{K_{l-1}}) = \text{argmax}_{a\in\{1,2,\ldots,l-1\}}[t^*(n, k_l, \boldsymbol{K_{l-1}}; a)]$.

Then, $E_l(n; t, k_l, \boldsymbol{K_{l-1}}) \leq E_l\left(n-1; t_l^*(n, k_l, \boldsymbol{K_{l-1}}), k_l, \boldsymbol{K_{l-1}} - \boldsymbol{e}_{a_l^*(n,k_l,\boldsymbol{K_{l-1}})}\right) + V_{a_l^*(n,k_l,\boldsymbol{K_{l-1}})}$, for

$t \le t_l^*(n, k_l, \boldsymbol{K_{l-1}})$ and $E_l(n; t, k_l, \boldsymbol{K_{l-1}}) > E_l\big(n-1; t_l^*(n, k_l, \boldsymbol{K_{l-1}}), k_l, \boldsymbol{K_{l-1}} - \boldsymbol{e}_{a_l^*(n,k_l,\boldsymbol{K_{l-1}})}\big) + V_{a_l^*(n,k_l,\boldsymbol{K_{l-1}})}$, for $t > t_l^*(n, k_l, \boldsymbol{K_{l-1}})$ because $E_l(n; t, k_l, \boldsymbol{K_{l-1}})$ is non-decreasing in *t*.

It means that every state, $(n, k_l, \boldsymbol{K_{l-1}})$, has its threshold time, $t_l^*(n, k_l, \boldsymbol{K_{l-1}})$, where the acceptance decision rule changes from "accept only (newly arriving) *l*-type items" to "accept one $a_l^*(n, k_l, \boldsymbol{K_{l-1}})$-type item in the system". We define the optimal decision rule to solve the problem using this threshold type optimal policy (see Figure 4-1).

### 4.4. Numerical Example: Problems with Two Item Types (*l*=2 and $k_2$ =0)

$$E_2(n; t, 0, \boldsymbol{K_1}) = nV_2 - \sum_{i=0}^{n-1}\left((n-i)V_2 - \sum_{j=0}^{k_1}\min(\lambda_1 t + k_1 - j, n-i)V_1 \frac{k_1!}{j!\,(k_1-j)!}(d_1)^j(1-d_1)^{k_1-j}\right)\frac{(\lambda_2 t)^i e^{-\lambda_2 t}}{i!} \quad (3)$$

From Equation (2), we define Equation (3). Figure 4-2 shows examples of problem with *l*=2 (i.e., problems with two item types). Figure (a) represents a problem with *n*=4 and $k_1$=3. Let's assume that that no item arrived or departed during the decision horizon. Then, the optimal decision was to accept one 1-type item at each threshold time, *t3*, *t2*, and *t1*, respectively. Figure (b) represents the case that threshold time was changed due to the new arrival of 1-type item. When $k_1$=1 (i.e. there was only one 1-type item in the system), threshold time was *t2*. However, after one 1-type item arrived, (i.e. $k_1$=2), threshold time became *t1*, which is smaller than *t2*. Figure 4-2 also shows that the maximum expected value is non-decreasing in *k* (c) and *n* (d) and non-increasing in *d* (e).

Using the same logic, we can define the traditional stochastic multiple-secretary problem, where all decisions are made immediately and any decision cannot be deferred or recalled. It has also a threshold type optimal policy, where each item type has its threshold time. The following equation represents the maximum expected value which can be obtained by selecting only *l*-type items at remaining time *t*.

$$E_l(n;t) = nV_l - \sum_{i=0}^{n-1}\left\{\left((n-i)V_l - E_{l-1}(n-i;t)\right)\frac{(\lambda_l t)^i e^{-\lambda_l t}}{i!}\right\} \quad (4)$$

*Basic decision rule*: At a given time *t*, if a newly arrived item, of which type is *a*, satisfies $E_l(n;t) \leq [E_l(n-1;t) + V_{l-a})]$, accept the item. Otherwise, do not accept it.

Figure 4-2-(f) shows the benefit of deferred decisions. Because decisions can be made as long as the items are still available (i.e., staying in the system), the deferred decision model should be more beneficial than the traditional model where decisions are made immediately.

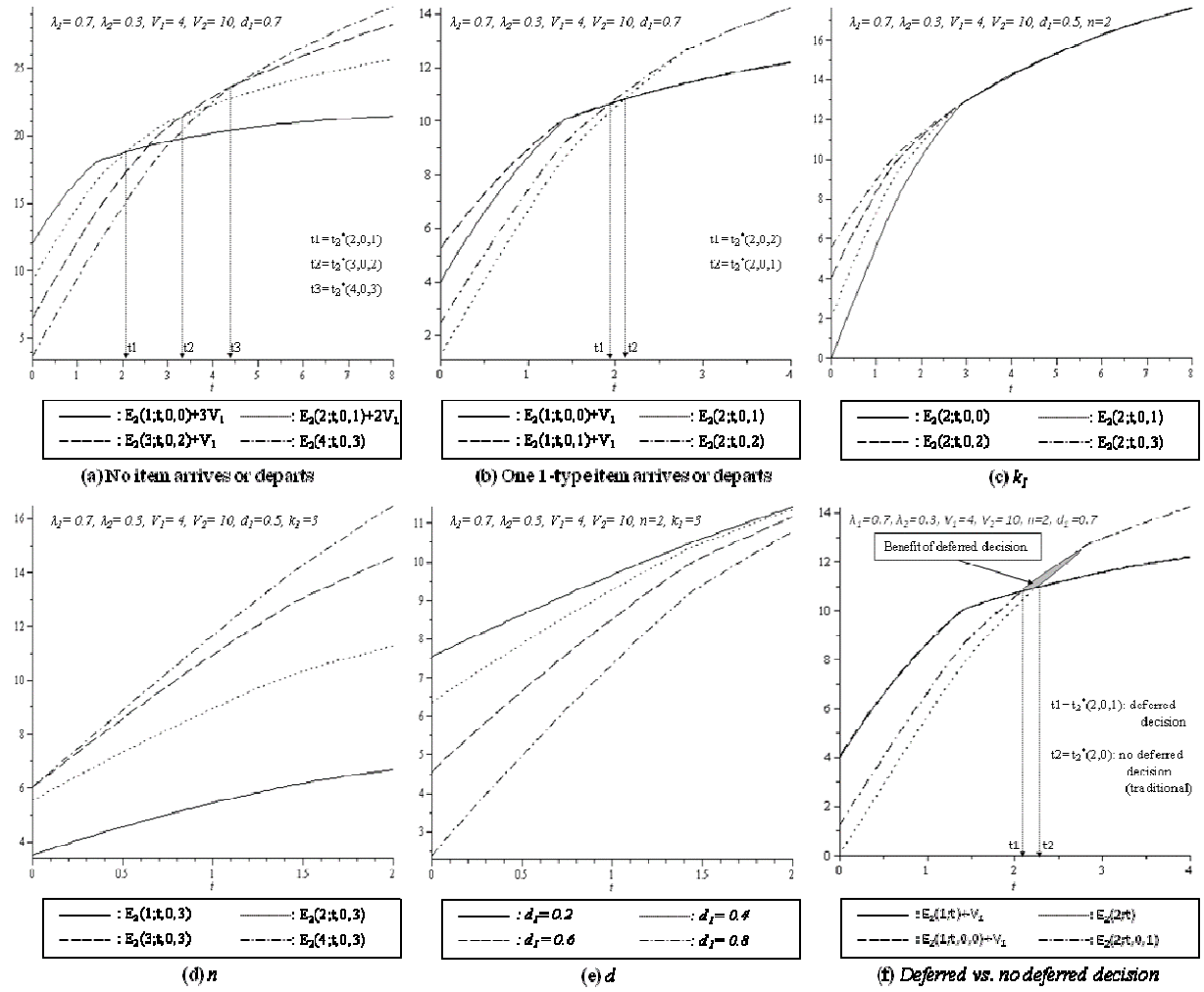


**Figure 4-2. Problems with *l*=2**

## 4.5. Conclusion and Discussion

Traditional secretary problems or online Knapsack problems strictly assume the instantaneous decisions; however, we observe that, in a number of practical problems such as software development outsourcing project selection problem of an IT service company, this assumption may be too strict. Therefore, we study a new version of stochastic multiple-secretary problem applying the concept of deferred decisions. The new problem allows a decision to accept or not a newly arrived item to be deferred to some extent. We show that the optimal policy for this problem is a threshold type policy and provide the optimal decision rule to solve the problem. Lastly, we show some numerical examples for the problem with two

types of items. One limitation of this research is the computational complexity to find the optimal policy. Therefore, developing heuristics or approximation algorithms for this problem would be the potential future directions.

# CHAPTER 5. Conclusion

This research investigates software project staffing problems. In the first two studies, we view the software development from a knowledge perspective and empirically examine the impacts of individual experience of project team members on a project team's performance. In the third study, we analytically address the software project selection problem.

What type of knowledge, among domain, technology, and methodology knowledge, is most influential to the performance of software development? In the first study, 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.

Software professionals can be classified as two types – specialists, those with deep / focused experiences within a particular domain, and generalists, who have experiences with a broad range of domains. Which type is most valuable to software projects is still an open empirical question. In the second study, we find that the impact of the type of professionals' experiences (i.e., specialist vs. generalist) varies by the type of software development knowledge required (i.e., domain, technology, and methodology) and by role of the software professional within the project (i.e., project managers vs. non-project managers).

In the third study, we define the software project selection problem of an IT service company as the *stochastic multiple-secretary problem with deferred decisions*. Unlike the traditional secretary (or online Knapsack) problem which strictly assumes that the decision to accept or not a newly arrived item needs to be made immediately upon its arrival, in software development project selection problem, the decision can be deferred to some extent. We find a threshold type optimal policy for this problem.

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. In addition, both empirical and analytical perspectives enable us to provide a full-fledged solution to project staffing problems.

## 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.

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

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.

Cohen, W.M. and Levinthal, D.A. (1990) "Absorptive capacity: a new perspective on learning and innovation," *Administrative Science Quarterly* (35:1) 128-152.

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.

Ferguson, T. S. (1989). Who solved the secretary problem? *Statistical Science*, 4(3), 282-289.

Freeman, P. R. (1983). The secretary problem and its extensions: A review. *International Statistical Review,* 51(2), 189-206.

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.

Hansen, M.T., and Oetinger, B.V. (2001) "Introducing T-shaped managers: Knowledge management's next generation," *Harvard Business Review* (79: 3) 107-116.

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.

Kleywegt, A. J., & Papastavrou, J. D. (1998). The dynamic and stochastic knapsack problem. *Operations Research, 46*(1), 17-35.

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.

Levinthal, D.A. and March, J. G. (1993) "The myopia of learning," *Strategic management Journal* (3) 95-112.

Lueker, G. S. (1998). Average-case analysis of off-line and on-line Knapsack problems. *Journal of Algorithms, 29*(2), 277-305.

Marchettispaccamela, A., & Vercellis, C. (1995). Stochastic online Knapsack-problems. *Mathematical Programming, 68*(1), 73-104.

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.

Papastavrou, J. D., Rajagopalan, S., & Kleywegt, A. J. (1996). The dynamic and stochastic knapsack problem with deadlines. *Management Science, 42*(12), 1706-1718.

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.

Ross, K. W., & Tsang, D. H. K. (1989). The Stochastic Knapsack-Problem. *IEEE Transactions on Communications, 37*(7), 740-747.

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.

van Slyke, R., & Young, Y. (2000). Finite horizon stochastic knapsacks with applications to yield management. *Operations Research, 48*(1), 155-172.

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.