Developer Learning Dynamics in Open Source Software Projects: A Hidden Markov Model Analysis

Param Vir Singh • Nara Youn • Yong Tan

University of Washington Business School, Box 353200, Seattle, Washington 98195-3200
{psidhu • nyoun • ytan}@u.washington.edu

Abstract

This work proposes a dynamic model of developer learning in open source software (OSS) projects. A Hidden Markov Model (HMM) is proposed to explain how the code contribution behaviors of OSS developers change as their levels of knowledge on their projects increase. In this model, discrete hidden states represent the unobserved knowledge levels of developers, and their observed code contribution behaviors are modeled as state dependent. Developers' knowledge levels evolve as they learn about the projects over time. Two modes of learning are considered: learning-by-doing (code development) and learning through interactions with peers. The model is calibrated using data spanning six years for 25 OSS projects and 251 developers hosted at Sourceforge. The proposed model identifies three knowledge states (high, medium, and low) and estimates the impact of the two modes of learning on the transition of developers between the three knowledge states. The model results suggest that in the low knowledge state developers exhibit the greatest inertia, followed by those in the medium and high states. Both modes of learning are found to have varying impact across the three knowledge states. Interactions with peers appear to be an important source of learning for developers in all states. A developer in the low state learns only through participation in threads started by others. Prior code contribution and starting discussion by initiating threads do not impact the knowledge level of a developer in the low state. Initiating threads, participating in threads started by others, and prior code contributions have positive impacts on the knowledge level of a developer in the medium or high state and, hence, influence his long term code contribution behavior. Explanations for these varying impacts of learning activities on the transitions of developers between the three states are provided. We also find a lack of persistence of knowledge in all states. The HMM better describes the data than a latent class model which would suggests that the learning activities have a long term, dynamic impact, rather than an immediate, static impact on the code contribution behavior of a developer.

Key words: Open source software development, learning, communication, Hidden Markov Model, latent class model.

1. Introduction

"*From personal experience in the Linux project there are plenty of people who given a little help and a bit of confidence boosting will become one with the best. There are many who won't but enough that will. As an example of this claim the original author of the Linux IPv6 code used to sit on IRC from Portugal playing with a few basic ideas and asking questions. After we helped him figure some of the kernel internals he wrote probably 75% of the Linux IPv6 stack and was last seen working in the USA for CISCO.*" senior developer at Linux, Alan Cox (Cox 1998).

Open source software (OSS) developer's code contributions to a project may be influenced by the extent of his experience or knowledge of the source code and his relevant coding skills. Developers vary in their initial knowledge of the source code and coding skills when they join an OSS project for the first time, and over time they learn about the code architecture, the skills required, and the project itself.

The strengths of the OSS phenomena have been closely associated with the evolution of novice or average skilled developers through interactions with sophisticated peers (Cox 1998, Raymond 1998, Singh et al 2006, Weber 2003). Various projects use several platforms for providing support to developers such as: developer mailing lists, discussion forums, and Internet relay chat (IRC) rooms, and etc. Developers use these communication channels for coordinating the code development effort as well as seeking help from peers (Cox 1998, Weber 2003). Besides the code development activities, knowledge exchanges over these communication networks are an important source of learning for a developer (Weber 2003). OSS development is a non-routine, complex and un-structured task (Singh and Tan 2005, Weber 2004) and, hence, support from experienced developers may go a long way in enhancing the understanding of a developer.

This research aims to explain the mechanisms through which developer-project interactions enhance a developer's knowledge level or improve his skill sets and, hence, influence his code contribution behavior. The wide availability of development data from OSS projects allows us to build models that will help explain development dynamics and provide insights for managerial initiatives. The key research questions addressed in this study are as follows:

- How does a developer evolve/learn over time? What activities enhance his/her knowledge levels?
- Are the impacts of learning activities homogeneous across developers? Can we identify activities at different levels of a developer's knowledge that can accelerate his rate of learning?

A developer in an OSS project possesses a certain level of knowledge of the source code, code-architecture or -design, relevant coding skills, and other issues related to the project. The majority of modules in an OSS project are developed by several developers. Hence, by participating in interactions with other developers, a developer may enhance his understanding of the project environment, the skills required, and the roles or tasks assigned. Such interaction may reduce duplication of effort and also provide a help-seeking and -providing platform for developers. A developer may also enhance his understanding of the project architecture and hone his coding skills by developing code. Conversely, the knowledge possessed by a developer may become irrelevant due to new advancements or depreciate due to forgetting as a result of inactivity on the part of a developer. In this work we consider the participation in code development and interactions with other developers as sources of learning. The knowledge level of a developer relevant to the project may increase or decrease based on his level of involvement in learning activities. However, the knowledge level of a developer at any time, whether or not known to that developer, is not observed in the data. We can only observe the code contribution behavior of a developer across time. As the opening quotation suggests, the knowledge level of a developer may positively influence his code contribution to a project. With such knowledge, we propose a modeling framework which deciphers the learning dynamics through the code contribution behavior of a developer.

A Hidden Markov Model (HMM) is developed which relates the unobserved knowledge states to the observed code contribution behavior. We investigate the impact of learning activities that affect the transitions of a developer between these hidden knowledge states. To be sure that learning does in fact occur and that developers move across different knowledge states, we compare our HMM with a latent class model. The latent class model incorporates cross-sectional heterogeneity in developers by segmenting them into different classes but does not allow a developer to shift between classes, i.e. it does not allow for learning effects. Comparison of the HMM and the latent class model will tell us if developers' coding

behaviors vary due to cross-sectional differences in their knowledge levels or because of dynamic changes in their knowledge levels over time.

Two modes of learning are considered: learning-by-doing (code development) and learning through interactions with peers. The model is calibrated using data spanning six years for 25 OSS projects (251 developers) hosted at Sourceforge (See Section 5 for the details of Sourceforge). Using the proposed model three knowledge states (high, medium, and low) are identified, and the marginal impact of the modes of learning on the transition of a developer between the three knowledge states is estimated. In the low knowledge state, developers exhibit the greatest inertia followed by the medium and high states. Both modes of learning are found to have varying impact across the three knowledge states. Interactions with peers appear as an important source of learning for developers in all states. A developer in the low state is found to learn only through participation in threads started by others. Prior code contribution and starting discussion by initiating threads do not impact the knowledge level of a developer in the low state. Initiating threads, participating in threads started by others, and prior code contributions are found to positively impact the knowledge level of a developer in the medium or high state and, hence, influence his long term code contribution behavior. We provide explanation for the varying impact of learning activities on transitions between the three states. We also find lack of persistence of knowledge in all states. The HMM we develop is a better fit for our data than a latent class model; this suggests that initiating threads, participating in threads started by others, and prior code contributions have a long term, rather than an immediate, impact on the code contribution behavior of a developer.

The rest of the paper is organized as follows. In Section 2 we present the theoretical background. The HMM model is laid out in Section 3 followed by the latent class model in Section 4. Section 5 explains data collection methodology. The variable description and estimation procedure are presented in Sections 6 and 7 respectively. The results from the HMM model are discussed in Section 8. Section 9 offers concluding remarks and directions of future research.

2. Literature and Theory

In this section, we discuss previous research related to developers' propensities to contribute to OSS. The literature on learning (especially learning by doing and learning from peers) and HMM are also discussed.

2.1 Developer Knowledge Heterogeneity

The OSS phenomenon has attracted a wide variety of developers under its fold. The motivations of these developers have been attributed to intellectual curiosity (Boston Consultancy Group 2003, Raymond 1999), labor economics (Lerner and Tirole 2002, Roberts et al 2006), the need to improve one's own specific programming skills (Lakhani and Hippel 2003), promises of higher future earnings (Haruvy et al 2003), altruism (Bonaccorsi and Cristina 2004), and individual need for the software developed (Lakhani and Wolf 2003, Niedner et al 2000). These different motivations imply that the OSS developers vary significantly in their inherent knowledge levels and skill sets. For instance, a highly skilled developer may join an OSS project to enjoy reputation benefits by showing his programming prowess to sophisticated peers (Lerner and Tirole 2002). In contrast, a novice developer may join the same project to learn through interactions with advanced or better skilled developers (Lakhani and Hippel 2003, Lakhani and Wolf 2003). The knowledge level and skills set of a developer influence his code contribution behavior (Cox 1998, Raymond 1999, Singh et al 2006, Weber 2003). Hence, even motivated developers may be heterogeneous in their code contribution propensities.

2.2 Learning

Learning is the process of acquiring knowledge or skills through experience or interactions that causes a change of behavior that is persistent and specified, or allows an individual to formulate or revise a mental construct (Anzai and Simon 1979, Ellis 1965, Harlow 1949). An individual may learn from the experience of others or from his own direct experience (Anzai and Simon 1979, Ellis 1965, Van de Ven and Polley 1992). A developer may learn, i.e., enhance his knowledge level and improve his coding skills, through code development and interactions with other developers (Cox 1998, Weber 2003).

2.2.1 Learning by doing

Learning by doing involves the learning of procedural or declarative knowledge (Argote 1993). The presence of learning curves or learning by doing has been well documented in several industries (Argote and Epple 1990, Darr et al 1995, March and Olsen 1972, Van de Ven and Polley 1992). In innovative processes, learning occurs through an adaptive process (Van de Ven and Polley 1992). Such an adaptive process involves a person undertaking a course of action, the environment producing a result, and the person

updating his course of action to increase the propensity of achieving his goals (March and Olsen 1972, Van de Ven and Polley 1992).

Learning curves are used to describe processes where the time required to perform a task declines at a decreasing rate as experience with the task increases (Argote et al 1990). In Learning curve studies, learning is modeled as a function of cumulative experience of performing a particular task (Argote 1993, Argote and Epple 1990). As evidence of the presence of learning effects in software development, expertise in programming is known to produce an order of magnitude improvement in program efficiency (Brooks 1987). Prior code contributions may reduce the effort required by a developer to code by his introduction to datasets, relationships among data-items, algorithms, invocation of functions, code architecture, among others (Brooks 1987, Weber 2003). Prior code contributions may thus enhance the knowledge level of a developer and therefore reduce the cost (time) of performing the task (code contribution). This implies that prior code contributions may positively impact the code contribution behavior of an OSS developer.

2.2.2 Learning via Communication

This mode involves transfer of knowledge through interactions in social relationships (Borgatti and Cross 2003, Granovetter 1973). Social interactions have been found as the means of getting new information (Borgatti and Cross 2003, Brown and Duguid 1991, Granovetter 1973). The communication network may be used for learning how to do one's job (Brown and Duguid 1991), manage expertise or specialized skills and knowledge (Faraj and Sproull 2000), and acquire new information (Granovetter 1973). Interpersonal relationships have been found to be a significant determinant of adoption decisions across products and industries (Bhikhchandani et al 1992, Foster and Rosenzweig 1995, Martilla 1971).

There has been considerable research focusing on individual and group learning. Groups may form interactive information systems to utilize the learning of individual members, foster the development of shared mental models and, hence, are able to out-perform individuals in cognitive tasks (Graydon and Griffin 1996, Mooreland 1999, Schillings 2003). The contribution of individual learning on group performance depends on the degree to which the task requires extensive coordination and the skills of the individuals (Argote 1999). Psychology research posits that at the group level information may be distrib-

uted across individuals, and individual members may draw upon social cognition to solve problems (Larson and Christensen 1993).

Software development teams need to manage their skills and knowledge effectively through coordination (Faraj and Sproull 2000). Brown and Duguid (1991) theorize that expertise is context dependent and evolves through interactions. Epple et al (1996) find that knowledge acquired by a team can be transferred if it is embedded in technology or organizational structure. In the context of software development, Kraut and Streeter (1995) argue that effective communication can lead to better knowledge sharing and, hence, reduce cost overruns, software unresponsiveness, software unreliability, and expensive modification costs. The communication networks in OSS projects allow developers to share their experiences or knowledge. This knowledge sharing could be about variety of issues such as code development, bugs, feature requests, and patch management, to name a few. A developer may seek help over the forum or help resolve others' problems. Applying his efforts or knowledge to different but related problem domains may help a developer in developing a deeper cognitive understanding of both (Graydon and Griffin 1996, Schilling et al 2003). Lakhani and Hippel (2003) found that 98% of the effort expended by information providers in fact returns direct learning benefits to those providers on Apache field support systems. Therefore, participation in communication exchanges may have a positive impact on the knowledge level of a developer through resolution of his problems or by exposure to new information and, hence, a positive influence on the code contribution behavior.

2.2.3 Depreciation of Learning

Learning is a process that depends on experience and leads to long-term changes in behavior potential (Anzai and Simon 1979, Ellis 1965). Persistent reinforcement is required for sustaining the potential, or else it becomes increasingly shallow, and eventually lost in the individual (Anzai and Simon 1979, Ellis 1965, Harlow 1949). In industrial settings, Argote et al (1990) and Epple et al (1996) find that the knowledge acquired through production depreciates rapidly. In OSS development, knowledge acquired by a developer through learning by doing or by interaction with peers may be lost through developer forgetting, changes in code architecture by other developers, and the like. Inactivity for a long period of

time may result in a developer's depreciation of knowledge and, hence, negatively influence his code contribution behavior.

2.3 Application of Hidden Markov Model

There have been multiple applications of HMM. Scott et al (2005) adopt HMM to study the effectiveness of medicine on patients' health, while Netzer et al (2006) use HMM to examine the impact of relationship building activities on the gift contributions of university alumni. Scott and Hann (2005) apply HMM to click stream data to analyze online customers' browsing and purchasing behavior.

3. Hidden Markov Model

A HMM is a model of a stochastic process that can not be observed directly but, can only be viewed through another set of stochastic processes that produce a set of observations (Rabiner 1989). We propose a HMM of developer evolution/learning in an OSS development environment (see Figure 1 for a graphical representation of the proposed HMM).

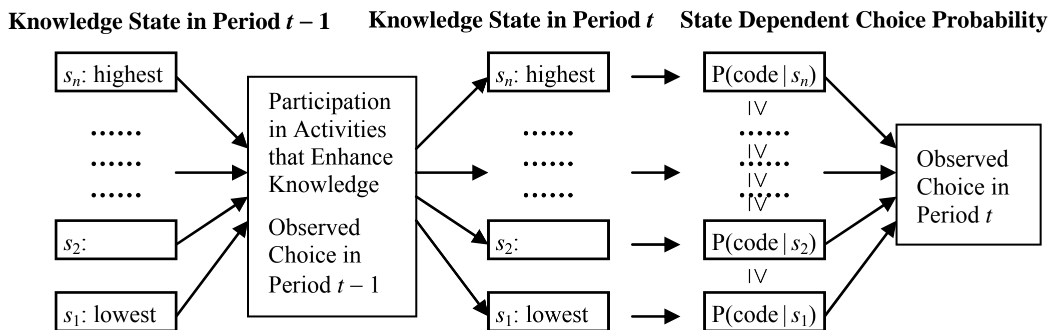

Figure 1: Hidden Markov Model of Developer Learning in OSS

Consider a set of developers who are involved in an OSS project. The entire time horizon is divided into periods starting from the inception of the project. For any given period, a developer resides in an unknown knowledge state. The transitions between knowledge states for a developer are affected by the observed learning activities in which a developer participates over a given period. The software coding effort of these developers at each time period constitutes the observed stochastic process. Probabilities of the particular choices are associated with unobservable knowledge states. The HMM, we develop, relates a developer's knowledge state and his coding effort and deciphers the hidden process of evolution/learning of a developer along the time horizon.

Given a set of knowledge states $s \in \{s_1, s_2, \cdots, s_n\}$, where s_1 is the lowest knowledge state and s_n is the highest knowledge state, and a sequence of observed choices $O = O_1 O_2 \cdots O_T$ ($O_t \in \{\text{code, no code}\}$ in our model), a HMM is comprised of three elements: (i) the initial state distribution, π, (ii) the state-transition probability distribution, Q, and (iii) the choice probability vector, A. A HMM requires the specification of n (number of states) and the specification of three probability measures (π, Q, A). For convenience, we use the following compact notation to represent the complete parameter set of the model:

$$\lambda = (\pi, Q, A).$$

Consider a fixed state sequence

$$S(i) = S_{i1} S_{i2} \cdots S_{iT},$$

where S_{i1} is the initial state for developer *i* and $S_{it} \in \{s_1, s_2, \cdots, s_n\}$, and an observation sequence $O(i)$ for developer *i*

$$O(i) = O_{i1} O_{i2} \cdots O_{iT},$$

where $O_{it} \in \{\text{code, no code}\}$. The probability of the observation sequence, $O(i)$, for the state sequence $S(i)$ and the parameter set λ is given by

$$P(O(i) \mid \lambda, S(i)) = \prod_{t=1}^{T} P(O_{it} \mid \lambda, S_{it}).$$

In HMM, any two adjacent observations are linked only through the hidden states. Thus we obtain,

$$P(O(i) \mid \lambda, S(i)) = a(S_{i1}, O_{i1}) \cdot a(S_{i2}, O_{i2}) \cdot \cdots \cdot a(S_{iT}, O_{iT}),$$

where $a(S_{it}, O_{it})$ is the probability of observing choice O_{it} given that developer i is in state S_{it} at time t. Note that $a(S_{it}, O_{it})$ is an element of the choice probability vector, $A(i,t)$. The probability of such a state sequence $S(i)$ is given as:

$$P(S(i) \mid \lambda) = \pi(i) q(S_{i1}, S_{i2}) \cdots q(S_{it}, S_{it+1}) \cdots q(S_{iT-1}, S_{iT}),$$

where $\pi(i)$ is the initial probability that developer i is in state S_{i1} in period $t = 1$. $q(S_{it}, S_{it+1})$ is the probability that developer i is in state S_{it+1} in period $t+1$ given he was in state S_{it} in period t. Note that

$q(S_{it}, S_{it+1})$ is an element of the state transition matrix $Q(i,t,t+1)$ for developer i. The probability that $O(i)$ and $S(i)$ occur simultaneously is given as:

$$P(O(i), S(i) | \lambda) = P(O(i) | \lambda, S(i)) P(S(i) | \lambda). \tag{1}$$

Hence, the probability of the observation sequence $O(i)$ given the model λ is also the likelihood of observing this sequence and is obtained by summing Equation (1) for all possible values of state sequence $S(i)$ (Rabiner 1989), explicitly,

$$L(O(i)) = P(O(i) | \lambda) = \sum_{\forall S(i)} P(O(i) | \lambda, S(i)) P(S(i) | \lambda).$$

The estimation of this likelihood by direct calculation is computationally quite expensive even for small values of T and n.[1] Following MacDonald and Zucchini (1997), the individual likelihood can be written in a more compact matrix notation as

$$L(O(i)) = \pi(i)\Lambda(i,1)Q(i,1,2)\Lambda(i,2)Q(i,2,3)\cdots Q(i,T-1,T)\Lambda(i,T)\mathbf{1}'.$$

Here, the matrix $\Lambda(i,t) = \text{diag}(a(s_1, O_{i1}), a(s_2, O_{i1}), \cdots, a(s_n, O_{i1}))$ and $\mathbf{1}$ is a $n \times 1$ vector of ones.

A developer's knowledge state in any period can be probabilistically recovered using the filtering approach (Hamilton 1989). The filtering approach utilizes only the information known up to time t to recover a developer's state in period t. The probability that a developer is in state s in period t is given as:

$$P(S_{it} = s | O_{i1}O_{i2}\cdots O_{it}) = \pi(i)\Lambda(i,1)Q(i,1,2)\Lambda(i,2)Q(i,2,3)\cdots Q_s(i,t-1,t)\Lambda(i,t) / L(O_{i1}O_{i2}\cdots O_{it}). \tag{2}$$

Here, $Q_s(i,t-1,t)$ represents the column of the transition matrix $Q(i,t-1,t)$ corresponding to the state s. $L(O_{i1}O_{i2}\cdots O_{it})$ is the likelihood of the observation sequence up to time t. To obtain the likelihood of the observation sequence $L(O(i))$, we need the model parameters Q, A and π. These parameters are defined in the following subsections.

---

[1] This can be computed quite efficiently by using forward part of the forward-backward procedure (see MacDonald and Zucchini (1997) Chapter 2, Baum and Egon (1967) or Baum and Sell (1968) for details of this procedure). However, the forward backward algorithm should be modified to incorporate the developer specific state dependent choice probabilities and transition matrices at time t. In fact, the calculation of the likelihood by the matrix notation is similar to the forward part of the forward-backward algorithm.

3.1 State Transition Matrix

The transition between states is modeled as a random walk where transitions to only the adjacent states are allowed. The random walk assumption keeps the model parsimonious. However, this assumption can be easily relaxed by assigning positive probabilities to transitions to non-adjacent states. This specification of the transitions to lower states is consistent with the psychology literature which states that interference from other tasks or delaying of performance causes forgetting. Hence, on resumption of the activity the performance is typically lower than when it was interrupted, but superior to when it initially began (Argote et al 1990, Kolers 1976). The random walk state transition matrix is defined as follows:

$$Q(i,t,t+1)=\begin{pmatrix} q(s_1,s_1) & q(s_1,s_2) & 0 & \cdots & 0 & 0 \\ q(s_2,s_1) & q(s_2,s_2) & q(s_2,s_3) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & q(s_n,s_{n-1}) & q(s_n,s_n) \end{pmatrix}.$$

Here, $q(s_j,s_k)=q(S_{it}=s_j,S_{it+1}=s_k)=P(S_{it+1}=s_k \mid S_{it}=s_j)$, and for each state $s_j, \sum_{k=1}^{n} q(s_j,s_k)=1$ and $0\le q(s_j,s_k)\le 1\ \forall\ j,k\in\{1,2,\cdots,n\}$.

This probabilistic transition is modeled by considering a propensity to transition which is affected by a developer's participations in learning activities such as interactions with other developers and his involvement in code development. A transition to a higher state occurs if the learning through these activities is higher than a certain high threshold value. Similarly, transition to a lower state occurs if this impact is lower than a certain low threshold value. Specifically, $q(S_{it},S_{it+1})$ is modeled as an ordered logit model:

$$q(s_j,s_{j+1})=1-\frac{\exp\left(\mu(h,s_j)-\beta_{s_j}R_{it}\right)}{1+\exp\left(\mu(h,s_j)-\beta_{s_j}R_{it}\right)},\ q(s_j,s_{j-1})=\frac{\exp\left(\mu(l,s_j)-\beta_{s_j}R_{it}\right)}{1+\exp\left(\mu(l,s_j)-\beta_{s_j}R_{it}\right)},\text{ and}$$

$$q(s_j,s_j)=\frac{\exp\left(\mu(h,s_j)-\beta_{s_j}R_{it}\right)}{1+\exp\left(\mu(h,s_j)-\beta_{s_j}R_{it}\right)}-\frac{\exp\left(\mu(l,s_j)-\beta_{s_j}R_{it}\right)}{1+\exp\left(\mu(l,s_j)-\beta_{s_j}R_{it}\right)},\ \forall\ j\in\{1,2,\cdots,n\}.$$

Here, $\mu(h,s_n)=\infty$, $\mu(l,s_1)=-\infty$, and R_{it} is a vector of variables that represent participation in learning activities by developer *i* in period *t*. The variables included in the vector R_{it} are explained in Section 6.1. Ideally, R_{it} should consist of variables that have continuing impact on the knowledge of a developer and, hence, a long term impact on his code contribution behavior. β_{s_j} is a parameter vector for impact of developer learning activities, R_{it}, on the propensity to transition from state s_j. The threshold value to move to an upper state is represented by $\mu(h,s_j)$, whereas the threshold value to move to a lower state is represented by $\mu(l,s_j)$. The constraint $\mu(h,s_j)>\mu(l,s_j)$ is applied to ensure that the high threshold is larger than the low threshold. Notice that in an intermediary state $s_j \in \{s_2,s_3,\cdots,s_{n-1}\}$, a developer has three options: (i) move up one state, (ii) stay in the same state, and (iii) move down one state. However, in the lowest state, s_1, a developer can either move up one state or stay in the same state. Similarly, in the highest state, s_n, a developer can either move down one state or stay in the same state.

3.2 State Dependent Choice Probability

The availability of development effort and overall project environment data allows us to model the choice probabilities as binary logit model. These choice probabilities are assumed to be state dependent. Specifically, these probabilities are modeled from the utility function of a developer as:

$$a\left(S_{it}=s_j,O_{it}=\text{code}\right)=\frac{\exp\left(\rho_{0j}+\rho_j W_{it}\right)}{1+\exp\left(\rho_{0j}+\rho_j W_{it}\right)},\ \forall\, j\in\{1,2,\cdots,n\}.$$

Here, given a developer i in state s_j, $a\left(S_{it}=s_j,O_{it}=\text{code}\right)$ represents the probability that he chooses to code in period *t*. Since we have a binary choice model, $\left(1-a\left(S_{it}=s_j,O_{it}=\text{code}\right)\right)$ represents his probability of choosing not to code in period *t*. The state specific coefficient ρ_{0j} represents a developer's intrinsic motivation to code. For identification of states, we impose a constraint $\rho_{0s_1}\le\rho_{0s_2}\le\cdots\le\rho_{0s_n}$.[2] This

[2] Note that in general this restriction is quite weak and may not ensure that the choice probabilities are non-decreasing in knowledge states as the ordering of only the intercept does not ensure final outcome probabilities. However, as the results show in Table 4, this milder restriction is sufficient to ensure that in our case choice

ensures that the choice probabilities are non-decreasing in learning states. The vector W_{it} is composed of time-varying covariates associated with the choice of individual *i* in period *t*. The parameter vector ρ_j represents the corresponding state specific coefficients. Ideally, W_{it} should consist of variables that have immediate impact on a developer's choice given his state. The variables that constitute W_{it} are explained in Section 6.2.

3.3 Initial State Distribution (π)

The model, we consider, is non-stationary and non-homogenous. Hence no stationary distribution exists for our model. Two options exist in this scenario: (i) generate a transition matrix at the mean of the covariates R_{it} and get the stationary distribution from this transition matrix, (ii) assume a stationary distribution based on intuition. Since the model is non-stationary and the length of the chain is very long (i.e. approx. 53 periods), the initial distribution does not have any impact on the final result. We follow the second option and assume an initial stationary distribution.

The parameter set to be estimated in the proposed HMM is dependent on the number of states *n*, and the number of variables in W_{it} and R_{it}. Specifically for $n > 1$, we need to estimate $\left((w+1)\times n + r\times n + (n-1)\times 2\right)$ parameters, where w is the column length of W_{it} and r is the column length of R_{it}. Out of the above set, $(w+1)\times n$ are the state dependent choice probability parameters associated with covariates W_{it}, and $r\times n$ are the transition probability parameters associated with the covariates R_{it}. The remaining $(n-1)\times 2$ are the threshold parameters for each state. For $n=1$, the only parameters needed to be estimated are the ones associated with the covariates W_{it}.

4. Latent Class Model

The latent class model posits that the impact of observed attributes on a developer's code contributions depends on latent heterogeneity among developers (Greene and Hensher 2003, Heckman and Singer

---

probabilities are non-decreasing in knowledge states. A much tighter condition that would ensure for certain that the choice probabilities are non-decreasing in knowledge states requires the restriction, $\rho_{0s_1} \le \rho_{0s_2} \le \cdots \le \rho_{0s_n}$, and that the vector of covariates W_{it} is mean centered.

1984). Following Greene and Hensher (2003), we assume that developers are sorted into a discrete set of N Classes. Let u_{it} be the utility for developer i, who belongs to class c, if he develops code in period t. This can be represented as:

$$u_{it} = \gamma_c X_{it} + \varepsilon_{it},$$

where X is a vector of covariates that impact the utility of code development for a developer. Parameter vector γ_c represents the corresponding vector of impact of the covariates on the utility of a developer in class c, where $c \in \{c_1, c_2, \cdots, c_N\}$. The central choice behavior is modeled as a binary logit model where an individual faces this binary choice in $t \in \{1, T\}$ choice situations. Let $p_{it|c} = \text{prob}(y_{it} = \text{code}|\text{class} = c)$ be the probability that developer i develops code $(y_{it} = \text{code})$ at time t, in class c. We have

$$p_{it|c} = \frac{\exp(\gamma_c X_{it})}{1 + \exp(\gamma_c X_{it})}, \ \forall \ c \in \{c_1, c_2, \cdots, c_N\}.$$

Given a class assignment, the contribution of developer i to the likelihood function is given as the joint probability of the sequence of his choice (whether to code or not), $y_i = (y_{i1}, y_{i2}, \cdots, y_{iT})$. Explicitly

$$p_{i|c} = \prod_{t=1}^{T} p_{it|c}.$$

The class assignment is unknown. Let π_{ic} be the prior probability for class c for developer i. Then, π_{ic} can be represented as follows:

$$\pi_{ic} = \frac{\exp(\tau_c C_i)}{\sum_c \exp(\tau_c C_i)}, \ \forall \ c \in \{c_1, c_2, \cdots, c_N\},$$

where $\tau_{c_N} = 0$ for identification purposes. C_i denotes the observable factors that enter the model for class membership. We assume that C_i includes only a constant term. This corresponds to a situation where no covariates are available for class membership.

The likelihood for individual i is the expectation (over all classes) of the class specific contributions:

$$p_i = \sum_c \pi_{ic} \, p_{i|c}.$$

The log likelihood function is represented as follows:

$$\ln L = \sum_{i=1}^{n} p_i = \sum_{i=1}^{n} \ln\left(\sum_{c} \pi_{ic} \prod_{t=1}^{T} p_{it|c} \right).$$

With the parameter estimates, the posterior estimates of the class probability can be achieved as follows:

$$\hat{\pi}_{c|i} = \frac{\hat{p}_{i|c}\hat{\pi}_{ic}}{\sum_{c} \hat{p}_{i|c}\hat{\pi}_{ic}}.$$

A strictly empirical estimator of the latent class within which a developer i resides would be that associated with the maximum value of $\hat{\pi}_{c|i}$.

5. Data Collection

Data was collected from the projects hosted at Sourceforge.net (SF) (Madey 2006). SF is the world's largest OSS project repository. Currently, it has more than 1,000,000 registered users and more than 100,000 projects. It provides a good sample of the OSS community to study the underlying dynamics. We consider only those projects that were registered at Sourceforge during the six month period from January 1, 2000 to June 30, 2000. There are 5,678 projects that fall into this category. We screened our sample to only include projects that had more than 10 developers[3] on January 1, 2006. The projects once hosted at Sourceforge are rarely removed even when the activity is negligible. Also, developers once registered in a project are rarely removed. This screening process reduced our sample size to 183 projects. We further screened the projects for developer email archives, Concurrent Versioning Systems (CVS), CVS commit power and number of files released since registering at Sourceforge. Some of the projects use other platforms for discussion rather than the mailing lists at Sourceforge. Each of the projects maintains a homepage where such information can be accessed. We visited the outside homepage of the projects to check if the developer mailing list at Sourceforge is the only mentioned source of developer discussion. Due to

[3] A lot of developers who register for a project do not contribute to the project at all. However, the actual code contribution of a registered developer can be known only by analyzing the CVS repository. Since we want to capture the learning through coding and interactions among developers, the restriction of 10 registered developers ensures us that there will be at least a few developers who make significant code contributions as well as a rich discussion list. For instance, one of the projects in the final sample has only 2 developers who made significant contributions but had 15 registered developers.

difficulty in matching the identity of developers at the two places (Sourceforge and the homepage), we do not consider such projects which have outside mailing lists. We also do not consider the projects which also use IRC channel along with the mailing list for developer discussion. In some projects only a selected few are given the power to commit to the CVS. This hampers the tracking of development activity. For the projects that we consider here all the registered developers have been granted the CVS commit power. In the final sample we have 25 projects. All of these projects were active on Dec 31, 2005. We collected the data from CVS repositories and mail list archives for these projects for the period spanning from the registration date of each project until Dec 30, 2005. This process provides us approximately 6 years of data for each project.

The source of the development effort data is the CVS log files that are used by all the projects under consideration. The CVS contains the current state of the project as well as the previous versions of the code. The data about participation of developers in a project is extracted through the CVS log entries. The CVS logs contain information about the changes (number of lines added or subtracted), date of change and the name of the developer who made changes for each source file[4]. TortoiseCVS was used to access the CVS repositories hosted at Sourceforge. From the description of the bug fix reports at Sourceforge we found that Sourceforge bug fix reports do not indicate the true fixer. Most of the times it is the project administrator who replies to bug submitters. Moreover, even for fixing the bugs any modification in the software source code is recorded in the CVS log files which we are already considering. Hence we do not consider bug fixing as a contributing activity in this model and focus only on code contributions.

The communication data was collected from the archives of the developer mailing lists[5] by using web agents. In mailing lists hosted at Sourceforge the emails that belong to a same original post are assigned same thread ID. This helped us in retrieving the information about the starter of a thread and subsequent participants in the thread. The date, time, thread starter and participants were recorded. Sometimes developers used different ID tags or names when communicating or committing in CVS. Names/IDs of all the CVS committers and communicators were scanned and matched for each project by

[4] CVS logs do not show the number of lines coded for initial release of a file. The number of coded lines for the initial version of a file is obtained by subtracting the changes made from the current version of a file.

human effort. The email information we use is based on threaded correspondences because that ensures that the original post was not just an announcement and demanded a reply. Project related data was collected from the project homepages at Sourceforge.net. The CVS commit and developer mailing list data for developers who had at-least 10 commits to the CVS is used to calibrate the HMM model. There were 251 developers who fit this criterion which provides us a rich enough dataset for HMM analysis. Mean statistics of the data collected are presented in Table 1.

Table 1: Mean Statistics of Data	
Overall Statistics	
Number of Projects	25
Number of Developers	251
Number of Choice Opportunities[6]	11320
Mean Number of Periods per Developer	53.3147
Length of a Period	1 month
Project Level Statistics	
Mean (Sourceforge) Rank of Projects	2796.1
Mean Number of Registered Developers Per Project	25.84
Mean Number of Developers Considered Per Project	10.04
CVS Statistics	
Mean Number of Periods of Commits Per Developer	16.5697
Mean Number of Total Lines Added Per Developer	15187.7
Developer Mailing List Statistics	
Mean Number of Threads Started by a Developer	35.426
Mean Number of Threads Participation by a Developer[7]	59.359
Mean Number of Threads Per Project[8]	692.9
Mean Number of Days a Thread spans[9]	5.14

6. Variable Descriptions

In this section, we describe the variables that constitute W_{it} and R_{it}.

6.1 Learning Encounters/Opportunities

These sets of variables include the interactions between developers as well as code development activities. These variables are hypothesized to have impact on the understanding/learning of a developer. These

[5] The mailing lists were combined for projects that had more than one mailing list for developers.

[6] This number does not include the observations that were lost due to incorporating the lagged cumulative number of lines coded.

[7] This only includes those threads which the developer did not start but participated in.

[8] This does not include Barrens Threads.

variables constitute the vector R_{it} used to calculate the state-transition probabilities. These are recorded after the first observation of a developer in the project. First observation of a developer is the first time he appears in the mailing list or the CVS commit logs.

We consider the variables that represent either of the two modes of learning for an OSS developer. Learning-by-doing studies have modeled learning as a function of cumulative experience of performing a task. In our OSS model, the learning-by-doing mode is represented by the amount of code accumulated to the previous time period. Brown and Duguid (1991) and Larson and Christensen (1993) theorize that expertise is context dependent and evolves through interactions. A developer may seek help from others on the mailing list. Other developers may provide help and the resulting interaction is arranged in a thread of emails. A thread usually serves the interests of a developer who started it by resolving his problem or through discussion of issues in which he may be interested. To capture this effect, we consider the number of threads started by a developer in the current period as a covariate for learning effect. A developer may also participate in threads started by someone else. This involvement on the part of a developer may be aimed at providing help to the knowledge seeker or seeking help himself. This might serve the interests of a developer directly or indirectly. A developer may gain a better understanding of his own issue by exerting his efforts in other but related issues (Graydon and Griffin 1996, Schilling et al. 2003). To capture this effect, we consider the number of threads that a developer did not start but in which he participated for the current period.

The variables that constitute vector R_{it} are as follows:[10]

$cumm_lines_committed_{it}$	=	(Cumulative number of source code lines committed by developer *i* till period $t-1$) / 100000.[11]
$inst_thread_start_{it}$	=	(Number of threads started by developer *i* in period t) / 10.
$inst_thread_part_{it}$	=	(Number of threads in which developer *i* participated but did not start in period t) / 10.

---

[9] Only about 20% of threads span more than 2 days.

[10] We also considered cumulative thread starts and thread participations as well as coding in prior period along with the above variables. However, these three variables were found to be insignificant for all the three states.

[11] All variables are properly rescaled to ensure solution stability.

6.2 Variables Impacting Choice

These variables influence the choice of a developer in each time period. They constitute the vector W_{it} used to calculate state dependent choice probabilities. These variables are hypothesized to have immediate impact on the choice variable (whether or not to code). These are also recorded after the first observation of a developer in the project. Lerner and Tirole (2002) argue that the immediate cost of code development to a developer consists of the opportunity cost of the time invested. The expected number of lines to be committed signifies the effort required of developers who choose to code. Lerner and Tirole (2002) and Roberts et al (2006) find that reputation concerns and the future returns associated with them are some of the primary reasons for developers to participate in OSS projects. Hence we hypothesize that a developer's decision to code or not would also be influenced by the project environment variables such as: overall development effort in the project, user base and activity in the community which would impact his returns. Sourceforge ranks the projects hosted on its website based on traffic, communication and development for each month.[12]

The variables that constitute vector W_{it} are as follows:

$explines_{it}$ = (Expected number of lines to be coded by developer i in period t [13]) / 10000.

$rank_{ipt}$ = (Sourceforge rank of project p in period $t-1$) / 100000.

6.3 Variables for the Latent Class Model

The latent class model does not allow for learning effects. Hence, the covariates vector, X_{it}, would consist of R_{it} and W_{it}. This implies that the covariates that are used to measure the learning effects in the HMM are used to measure the coding choice impact in the latent class models.

In the next section we describe the procedure to estimate the parameters associated with the variables defined in this section.

[12] http://sourceforge.net/docman/display_doc.php?docid=14040&group_id=1#project_stats_access

[13] It is estimated by number of lines that a developer last coded.

7. Estimation Procedure and Model Selection Criteria

One of the issues to be considered is how to choose the number of states N (in the latent class models) and n (in the HMM). Roeder et al (1999) and Greene and Hensher (2003) suggest the use of Bayesian Information Criterion (BIC) for model selection:

$$\text{BIC} = \ln L - size \times \ln(D)/2 .$$

Here *size* is the number of parameters in the model, and D is the number of developers. We consider seven scenarios: three scenarios each (two-, three-, four-state) for the Latent Class and HMM, and the one state[14] case. These scenarios are run separately and their log-likelihood values are obtained.

We first assume initial state distributions for these two-, three- and four- states as described in Table 2. All the three initial state distributions assign a high probability for a developer to be in state 1.

Table 2: Initial State Distribution

Two State Scenario	Three State Scenario	Four State Scenario
$\pi = \begin{bmatrix} 0.7 \\ 0.3 \end{bmatrix}$	$\pi = \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}$	$\pi = \begin{bmatrix} 0.6 \\ 0.2 \\ 0.15 \\ 0.05 \end{bmatrix}$

Maximum Likelihood Estimation is used for parameter estimation. We used sequential BFGS Newton-Raphson algorithm to maximize the likelihood. The log-likelihoods for all scenarios are shown in Table 3. The three-state HMM outperforms all other model specifications. The one-state model is outperformed by all the other models, which implies that there is significant heterogeneity among code contribution behavior of developers in OSS projects. Since the three-state HMM outperforms the latent class models, it shows that the prior code contributions and interactions with other developers have a long term rather than a short term impact on the code contribution behavior of a developer. This implies that a developer acquires knowledge or skills through experience or interactions that causes a change of behavior that is persistent.

[14] The one state model implies that the developers are homogeneous and does not account for any learning.

Table 3: Comparison of HMM and Latent Class Models				
Model	**Number of States**	**Log-Likelihood**	**BIC**	**Variables Estimated**
	One	–6633.0	–6649.6	6
HMM	Two	–5241.2	–5279.9	14
	Three	**–5113.6**	**–5174.4**	**22**
	Four	–5099.1	–5181.9	30
Latent Class	Two	–6022.3	–6058.1	13
	Three	–5679.8	–5735.0	20
	Four	–5695.7	–5770.2	27

8. Results and Discussion

The estimated parameters for the three-state model are shown in Table 4, where the corresponding standard errors are presented in parentheses.

Table 4: Estimated Parameters for the Three State HMM			
Parameter	**State 1**	**State 2**	**State 3**
ρ_{0s}	–3.669 (0.185)	–1.959 (0.179)	1.320 (0.063)
$\rho(explines)$	–0.627 (2.338)	1.203 (0.328)	45.281 (2.054)
$\rho(rank)$	–14.809 (1.293)	–9.628 (1.751)	–4.249 (0.801)
$\mu_{h,s}$	2.903 (0.135)	2.178 (0.049)	-----------------
$\mu_{l,s}$	----------------	–2.445 (0.135)	–0.700 (0.093)
$\beta(inst_thread_start)$	–1.052 (0.843)	11.999 (1.126)	4.155 (0.908)
$\beta(inst_thread_part)$	3.577 (0.637)	3.977 (0.820)	2.083 (0.441)
$\beta(cumm_lines_committed)$	–41.079 (21.16)	0.742 (0.220)	1.842 (0.380)

8.1 Knowledge State – Code Contribution Behavior Relationship

The identification of the states is based primarily on the state specific intrinsic propensity to code $(\rho_{01}, \rho_{02}, \rho_{03})$. The intrinsic probability of code development given state 1 is 2.49%, given state 2 it is 12.36%, and given state 3 it is 78.92%. Hence we can say that states 1, 2, and 3 represent states of "low," "medium," and "high" knowledge states, respectively.

The coefficients for *explines* are positive for the medium and high states, and negative but insignificant for the low state. This suggests that OSS developers enjoy coding rather than seeing it as effort. Though counter-intuitive, this finding fits nicely with the existing research, which has found that OSS developers are motivated by intellectual curiosity and find it pleasurable to code (Boston Consultancy

Group 2003, Singh et al 2006, Weber 2003). Also note that the coefficient is higher for the high state compared to the medium state, which implies that a developer in the high state enjoys coding more than a developer in the medium state.

The coefficients for *rank* are negative for all the three states; this implies that a developer is more likely to contribute code to a project when the project environmental variables are giving positive signals. That is, a developer is positively influenced to contribute code by the overall development effort in the project, user base and activity in the community which would impact his returns. Note that as the knowledge level of a developer increases he starts caring less about the *rank* of the project while making his code contribution decision. Hence, in the high state, a developer is not only highly intrinsically motivated to code but also enjoys coding and is less sensitive to the project environment.

8.2 Knowledge State Transitions

As expected, the threshold for moving to higher (lower) states are positive (negative). These thresholds represent the intrinsic propensity to transition to another state. Table 5 presents the intrinsic propensities to transition for a developer.

Table 5: Transition Matrix (Intrinsic Propensity to Transition)

$t \rightarrow t+1$	Low	Medium	High
Low	94.80%	5.20%	0%
Medium	7.98%	81.85%	10.17%
High	0%	33.12%	66.82%

The probability for a developer in the low state in period t to move to the medium state is only 5.20%. Similarly the probability for a developer in the medium state in period t to move to the high state is 10.17% and to move to the low state it is 7.98%. For a developer in the high knowledge state the probability is 33.12% to move to the medium state. Of all the three knowledge states, the lowest state is the "stickiest," i.e. a developer in the low state needs the maximum help to move up.

8.3 Learning through Communication Channel

The coefficients for $inst_thread_start$ are positive and significant for developers in the medium and high knowledge states but negative and insignificant for a developer in the low knowledge state. Also

note that the coefficient is the highest for the medium knowledge state. This implies that asking questions by starting threads helps a developer in the medium knowledge state the most. This increases the probability of transition to the high state from 10.17% to 86.83% for a developer in the medium state. This increase is highly significant in light of the observation that this transition increases the intrinsic propensity to code for a developer from 12.36% to 78.92% besides making him less sensitive to rank and him finding coding more pleasurable. However, a developer in the low knowledge state does not learn at all by starting threads. This may be due to the reason that a developer in the low knowledge state may not be able to frame his questions properly, which may lead to the other developers either not understanding the question or finding it not worthwhile to answer. Raymond and Moen's (2001) observation that average or below average users/developers do not spend time thinking about the problem before posting a question on mailing lists seems to be applicable to developers in the low state. They further state that expert developers "have a reputation for meeting simple questions with what looks like hostility or arrogance." Table 6 indicates the impact of participation in communication on the learning of a developer with average number of thread starts.

Table 6: Transition Matrix (Communication)[15]

Threads Starts				**Thread Participation**			
$t \to t+1$	Low	Medium	High	$t \to t+1$	Low	Medium	High
Low	94.80%	5.20%	0%	Low	79.90%	20.10%	0%
Medium	0.15%	13.02%	86.83%	Medium	1.49%	59.07%	39.44%
High	0%	9.73%	90.27%	High	0%	17.79%	82.21%

The coefficients for $inst_thread_part$ are positive as well as significant for all three knowledge states. It implies that a developer learns by participating in discussions started by others irrespective of his knowledge level. It fits well with Raymond and Moen's (2006) argument that good questions help developers gain their understanding, and often reveal problems they might not have noticed or thought about otherwise. It is also interesting to see that a developer in the low knowledge state learns only by participating in threads started by others. As Table 6 shows, the probability of transition for a developer in the

[15] This matrix is obtained by assuming the *cum_lines_committed* to be 50% of the average number of lines committed per developer. The mean number of *inst_thread_start* (*inst_thread_part*) was obtained by averaging the total number of *inst_thread_start* (*inst_thread_part*) by the total number of periods with positive *inst_thread_start* (*inst_thread_part*).

low knowledge state to move to the medium knowledge state increases from 5.20% to 20.10%. This increase is quite significant given the "stickiness" of the low state. Another interesting and important effect of participation in communication for a developer in the medium or high state is that it reduces the probability for transitioning to a lower state quite considerably.

8.4 Learning by Doing

The coefficients for $cumm_lines_committed$ are positive and significant for developers in the medium and high knowledge states but negative and insignificant for a developer in the low knowledge state. Table 7 shows the transition matrix for the impact of code development on the learning of a developer.

Table 7: Transition Matrix (50% cum dev)

$t \to t+1$	Low	Medium	High
Low	94.80%	5.20%	0%
Medium	7.58%	81.72%	10.70%
High	0%	30.16%	69.84%

This transition matrix is obtained by considering a situation where a developer has achieved 50% of the average total coding in period $t-1$ and did not participate in any coding or interaction with other developers in period t. It is interesting to see that the impact of code development, though positive, is much less pronounced in both the medium and high knowledge states. This indicates that developers who contribute significantly are experienced in coding before joining the project. However, the positive impact of coding may be due to the learning of the architecture and design of the source code which is specific to a group.

Table 8: Transition Matrix (All Activities)

$t \to t+1$	Low	Medium	High
Low	79.90%	20.10%	0%
Medium	0.03%	2.68%	97.29%
High	0%	4.23%	95.77%

Table 8 shows the transition matrix for a developer when he participates in both types of communication and have achieved 50% code development. As expected participation in all the learning activities has a huge impact on moving developers or keeping developers in the high state. It is much more pro-

nounced for a developer in the medium knowledge state as he has a probability of 97.29% of moving to the high state.

8.5 Accumulation and Depreciation of Knowledge

The matrices in Table 9 are obtained using the *Chapman-Kolmogorov Equations*. If a developer in the low knowledge state actively participates in all learning activities at an average level continuously for 6 months, then he has a probability of 64.79% of being in the high state in the 6th month. For the medium and high knowledge states, the probabilities of being in the high state after 6 months of continuous participation in learning activities are 95.91% and 95.92%, respectively.

Table 9: Transition Matrix (6 step transition matrix)

Active for 6 months[16]				**Inactive for 6 month**[17]			
$t \rightarrow t+1$	Low	Medium	High	$t \rightarrow t+1$	Low	Medium	High
Low	26.02%	9.19%	64.79%	Low	77.15%	19.25%	4.09%
Medium	0.01%	4.08%	95.91%	Medium	28.06%	52.10%	19.92%
High	0%	4.08%	95.92%	High	16.79%	56.16%	27.09%

If a developer in the high state does not participate in any of the activities for 6 continuous months then he has a probability of 72.95% of moving to a lower state (specifically 16.79% for the low state and 56.16% for the medium state). Similarly, continuous inactivity for 6 months on the part of a developer in the medium state can lead to a probability of 28.06% of being in the low state. Note the probability of being in the high state is lower than that of being in the low state. This implies that if a developer does not participate in any learning activities then he may lag behind in his understanding of the code developed by others, as well as the issues discussed during that period, and might also unlearn a few coding tricks. This indicates the lack of persistence of learning as argued by Argote et al (1990) and Epple et al (1996). This finding is also consistent with the psychology literature on forgetting when interrupted or delayed (Argote et al 1990, Kolers 1976).

---

[16] Here the developer participates in an average level of activities for the continuous 6 months. The instantaneous variables, *inst_thread_start* and *inst_thread_part*, take their respective mean values. The cumulative variable, *cumm_lines_committed*, starts at 50% of the average number of lines committed per developer and increases at a rate of 284.9 lines per period (15187.7/53.3147, the average number of lines committed per developer per period).

[17] This transition matrix is the 6 step transition matrix from transition matrix in Table 7.

8.6 Posterior Analysis of Group and Individual Behaviors

In any given period, an individual developer can be classified into a specific state according to posterior probability calculation, as described in Equation (2). Figure 2(a) depicts the trend (over time) of the distribution of developers in three states, where the two curves plot the boundaries that separate the low, medium, and high states. While developers in the medium state remain at roughly 25%, there is a decline in high state developers towards the end of time horizon under study. This observation is confirmed in Figure 2(b) which shows that the average state drops after the maximum (a plateau from periods 15 to 30, approximately) is reached. A closer analysis of the individual projects shows that most of the projects under consideration achieved maturity/production stability around this period and, hence, the development activity after 30 periods is minimal.

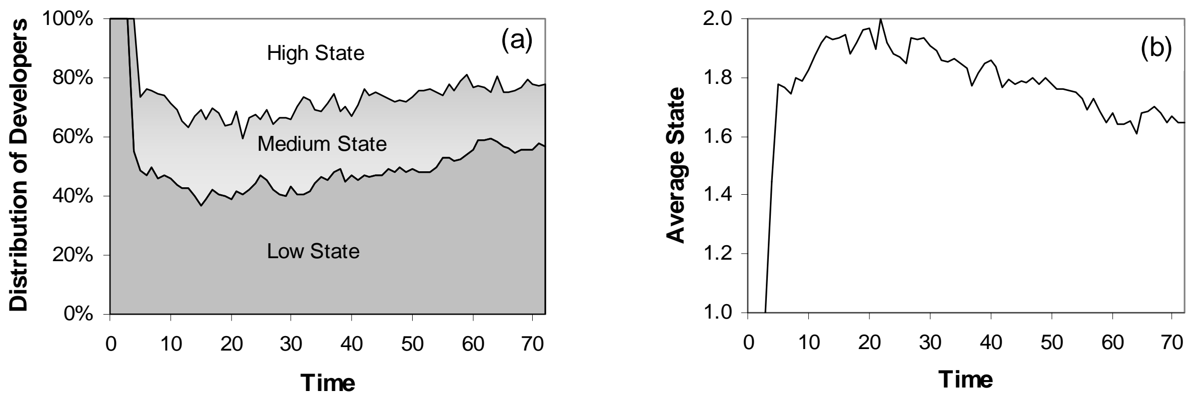

Figure 2: State Distribution of Developers (a) and Average State (b) against Time

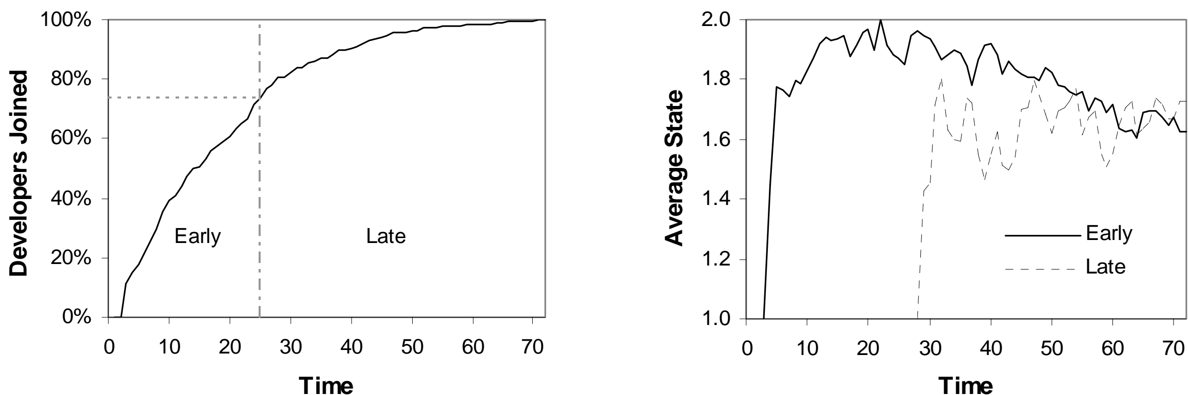

Figure 3: Developers Joined (in Percentage) **Figure 4:** Average State for Early or Late Joiners

As developers join in a project at different time points, it can be argued that the ones who join late may have caused this drop. In Figure 3, we plot developers joined (in percentage of all 251 developers) as

a function of time. By period 25, roughly one third of the time horizon, 75% have already joined, whom we classify as "early" joiners, and the rest "late" ones. Late joiners do have a lower average state which remains more or less the same. Early joiners are able to achieve a higher average state which, however, can not be sustained. This drop may again be a result of minimal development activity after project maturity.

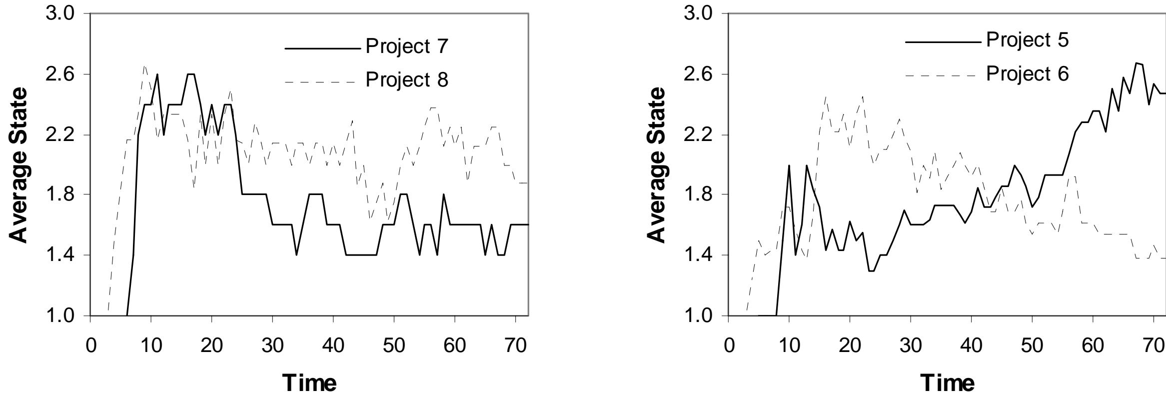

Figure 5: Average State for Developers in a Project

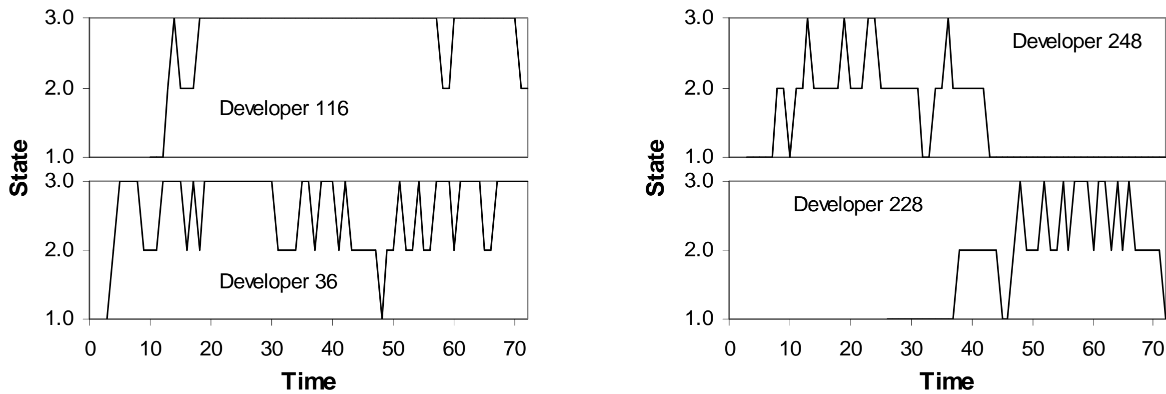

Figure 6: State of Individual Developer against Time

Each project may have its own characteristics. As Figure 5 shows, developers in Project 8 maintain the same level of knowledge state, while Project 7 has two distinct levels. Project 7 is a small project that achieved project maturity in a few months since its inception at Sourceforge. Whereas Project 8 has not achieved project maturity till the end of data collection period. Project 5 gives more learning opportunities, but Project 6 appears to behave in opposite way. Compared to Project 6, we witnessed intense communication among developers as the time elapsed in Project 5. At the individual level (Figure 6), we find various types of behaviors, for example dedicated (developer 116), on-and-off (developer 36), late joiner

(developer 228), and early joiner who stops contributing (developer 248). One interesting observation that can be made from Figure 6 is that once a developer leaves the low state he rarely returns back to it.

9. Conclusions and Future Research Directions

In this paper, we develop a Hidden Markov Model of developer learning dynamics in OSS projects. Discrete hidden states are introduced to represent the unobserved knowledge levels of developers, and their observed code contribution behaviors are modeled as state dependent. Developers' knowledge levels evolve as they learn about the projects over time. We consider two modes of learning: learning-by-doing (code development) and learning through interactions (communications) with peers. Publicly available data from CVS repositories and mailing lists is used for model estimation. This allows us to identify three knowledge states which can be related to developers' intrinsic propensity to code. We also examine the impacts of the two modes of learning on the transition of developers between these knowledge states. Furthermore, individual developers are classified into respective knowledge states applying posterior probability calculation, and their dynamic learning patterns are discussed at project and individual levels.

The main contributions of this paper are two-fold. We have advanced research in OSS development by applying a model, and using data from publicly available CVS repositories and discussion lists, to investigate developer learning which in turn impacts code development. We have also enriched the individual/organizational learning literature by proposing a Markovian framework for evaluating individual learning dynamics. The findings of this work have several important managerial implications. First, this study establishes a number of knowledge states for OSS developers, and proposes ways for classification of developers. This makes possible to examine methods by which a manager can alter a developer's knowledge level and affect long term contribution behavior. Second, we find that even motivated OSS developers are heterogeneous in their code-contribution behavior. Developers in the high knowledge state are more likely to contribute code. Hence, there is incentive for a manager to encourage active learning. Third, a developer's interactions with the project enhances his knowledge and, hence, the likelihood of code contribution. Two modes of learning (by doing or via communication) are found to have varying impact across knowledge states. Developer interactions are identified as the main source of learning for

developers in any state. This suggests that a manager or developer in the high state should foster richer discussion in group mailing lists. Finally, this study suggests that the learning activities have a long term, dynamic, rather than an immediate, static impact on the code contribution behavior of a developer.

This model can also be applied to understand the dynamics of learning in more structured software development projects such as proprietary software development. Though we consider only a binary choice variable, the model can be easily modified to consider a multinomial choice variable. Similarly the transitions could be allowed between non-adjacent states. It would be interesting to see whether these increased model complexities yield additional managerial insights.

References

Anzai, Y., H. Simon. 1979. The Theory of Learning by Doing. *Psychological Review* **86**(2) 124–140.

Argote, L. 1993. Group and Organizational Learning Curves: Individual, System and Environment Components. *British Journal of Social Psychology* **32** 31–51.

Argote, L., S.L. Beckman, D. Epple. 1990. The Persistence and Transfer of Learning in Industrial Settings. *Management Science* **26**(2) 140–154.

Baum, L.E., J.A. Egon. 1967. An Equality with Applications to Statistical Estimation for Probabilistic Functions of a Markov Process and to a Model of Ecology. *Bulletin of American Meteorological Society* **73** 360–367.

Baum, L.E., G.R. Sell. 1968. Growth Functions for Transformations on Manifolds. *Pacific Journal of Mathematic* **27**(2) 211–227.

Bhikhchandani, S., D. Hirschleifer, I. Welch. 1992. A Theory of Fads, Fashion, Custom and Cultural Change as Informational Cascades. *Journal of Political Economy* **100**(5) 992–1026.

Bonaccorsi, A., R. Cristina. 2004. Altruistic Individuals, Selfish Firms? The Structure of Motivation in Open Source Software. *First Monday* **9**(1). Available at http://firstmonday.org

Borgatti, S.P., R. Cross. 2003. A Relational View of Information Seeking and Learning in Social Networks. *Management Science* 49(4) 432–445.

Boston Consultancy Group. 2002. Boston Consulting Group/OSDN Hacker Survey. Retrieved September 1, 2005 at http://www.ostg.com/bcg/.

Brooks, F.P. 1987. No Silver Bullet, Essence and Accidents of Software Engineering. *Computer* **20**(4) 10–19.

Brown, J.S., P. Duguid. 1991. Organizational Learning and Communities of Practice: Towards a Unifying View of Working, Learning and Innovation. *Organization Science* **2**(1) 40–57.

Cox, A. 1998. Cathedrals, Bazaars and the Town Council. *Slashdot.Org*, Retrieved October 1, 2004 at http://slashdot.org/features/98/10/13/1423253.shtml.

Darr, E.D., L. Argote, D. Epple. 1995. The Acquisition, Transfer, and Depreciation of Knowledge in Service Organizations: Productivity in Franchises. *Management Science* **41**(11) 1750–1762.

Ellis, H. 1965. *The Transfer of Learning*. Macmillan Company, New York.

Epple, D., L. Argote, K. Murphy. 1996. An Empirical Investigation of the Microstructure of Knowledge Acquisition and Transfer through Learning by Doing. *Operations Research* **44**(1) 77–86.

Faraj, S., L. Sproull. 2000. Coordinating Expertise in Software Development Teams. *Management Science* **46**(12) 1554–1568.

Foster, A. D., M. R. Rosenzweig.1995. Learning by Doing and Learning from Others: Human Capital and Technical Change in Agriculture. *The Journal of Political Economy* (**103**)6 1176–1209.

Granovetter, M. 1973. The strength of weak ties. *American Journal of Sociology* **78** 1360–1380.

Graydon, J., M. Griffin. 1996. Specificity and Variability of Practice with Young Children. *Perceptual and Motor Skills* **83** 83–88.

Greene, W.H. 2003. *Econometric Analysi*s, 5th Edition. Prentice Hall, Upper Saddle River, New Jersey.

Greene, W.H., D.A. Hensher. 2003. A Latent Class Model of Discrete Choice Analysis: Contrasts with Mixed Logit. *Transportation Research* **B**.

Hamilton, J.D. 1989. A New Approach to the Economic Analysis of Non-stationary Time Series and the Business Cycle. *Econometrica* **57**(2) 357–384.

Harlow, H.E. 1949. The Formation of Learning Sets. *Psychological Review* **56** 51–65.

Haruvy, E.E., F. Wu, S. Chakravarty. 2003. Incentives for Developers' Contributions and Product Performance Metrics in Open Source Development: An Empirical Investigation. *University of Texas at Dallas Working Paper.*

Heckman, J., B. Singer. 1984. Econometric Duration Analysis. *Journal of Econometrics* **24** 63–132.

Kraut, R.E., L.A. Streeter. 1995. Coordinating in Software Development. *Communications of the ACM* **38**(3) 69–81.

Lakhani, K., E.V. Hippel. 2003. How Open Source Software Works: 'Free' User-to-User Assistance. *Research Policy* **32** 923–943.

Lakhani, K., B. Wolf. 2003. Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. *MIT Working Paper*. Available at http://papers.ssrn.com/sol3/papers.cfm?abstract_id=443040, accessed 1 November 2005.

Larson, J.R., C. Christensen. 1993. Groups as Problem-Solving Units: Towards a New Meaning of Social Cognition. *British Journal of Social Psychology* **32** 5–30.

Lerner, J., J. Tirole. 2002. Some Simple Economics of Open Source. *Journal of Industrial Economics* **50**(2) 197–234.

MacCormack, A., J. Rusnak, C. Baldwin. 2006. Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code. *Management Science* **52**(7) 1015–1031.

MacDonald, I.L., W. Zucchini. 1997. *Hidden Markov and Other Models for Discrete-Valued Time Series*. London: Chapman and Hall.

Madey, G. 2006. SourceForge Research Data Archive, Notre Dame, IN. http://zerlot.cse.nd.edu/mywiki/

March, J.G., J.P. Olsen. 1975. The Uncertainty of the Past: Organizational Learning under Ambiguity. *European Journal of Political Research* **3** 147–171.

Martilla, J.A. 1971. Word-of-mouth Communication in Industrial Adoption Process. *Journal of Marketing Research* **8** 173–178.

Moreland, R.L. 1999. Transactive memory: Learning who knows what in work groups and organizations. L. Thompson, J. Levine, & D. Messick (Eds.), *Shared cognition in organizations: The management of knowledge*. Mahwah, N.J., 3–31.

Netzer, O., J. Lattin, V.S. Srinivasan. 2005. A Hidden Markov Model of Customer Relationship Dynamics. Stanford GSB Research Paper No. 1904. Available at SSRN: http://ssrn.com/abstract=776765.

Niedner, S., G. Hertel, S. Hermann. 2000. Motivation in Open Source Projects: An Empirical Study Among Linux Developers. *Research Policy* **32**(7) 1159–1177.

Rabiner, L.R. 1989. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of IEEE* **77**(2) 257–285.

Raymond, E.S. 1998. The Cathedral and the Bazaar. *First Monday* **3**(3), Retrieved on December 1, 2005 at http://www.firstmonday.org.

Raymond, E.S., R. Moen. 2006. How to Ask Questions The Smart Way. Retrieved on March 16, 2006 at http://www.catb.org/~esr/faqs/smart-questions.html.

Roberts, J., I.H. Hann, S. Slaughter. 2006. Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects. *Management Science* **52**(7) 984–999.

Scott, S. L., G.M. James, C.A. Sugar. 2005. Hidden Markov Models for Longitudinal Comparisons. *Journal of the American Statistical Association* **100**(470) 359–369.

Scott, S.L., I.L. Hann. 2005. A Doubly Nested Hidden markov Model for Internet Browsing Behavior. *University of Southern California Working Paper*.

Shilling, M.A., P. Vidal, R.E. Ployhart, A. Marangoni. 2003. Learning by Doing Something Else: Variation, Relatedness, and the Learning Curve. *Management Science* **49**(1) 39–56.

Singh, P.V., Y. Tan. 2005. Stability and Efficiency of Communication Networks in Open Source Software Development. *Proceedings of the Fifteenth annual Workshop on Information Technology and Systems WITS, Las Vegas.*

Singh, P.V., Y. Tan, D. Dey. 2006. Stability and Efficiency of Communication Networks in Open Source Software Projects. *University of Washington Working Paper*.

Van de Ven, A.H., D. Polley. 1992. Learning While Innovating. *Organization Science* **3**(1) 92–116.