Abstract

This study examines whether developers learn from their experience and from interactions with peers in OSS projects. A Hidden Markov Model (HMM) is proposed that allows us to investigate (1) the extent to which OSS developers actually learn from their own experience and from interactions with peers, (2) whether a developer’s abilities to learn from these activities vary as she evolves/learns over time, and (3) to what extent developer learning persists over time. We calibrate the model on six years of detailed data collected from 251 developers working on 25 OSS projects hosted at Sourceforge. Using the HMM three latent learning states (high, medium, and low) are identified and the marginal impact of learning activities on moving the developer between these states is estimated. Our findings reveal different patterns of learning in different learning states. Learning from peers appears as the most important source of learning for developers across the three states. Developers in the medium learning state benefit most through discussions that they initiate. On the other hand, developers in the low and the high states benefit the most by participating in discussions started by others. While in the low state, developers depend entirely upon their peers to learn whereas when in medium or high state they can also draw upon their own experiences. Explanations for these varying impacts of learning activities on the transitions of developers between the three learning states are provided.

Key words: Open source software, learning, communication, Hidden Markov model

--------------------

We thank Ramayya Krishnan, Prabuddha De, Tridas Mukhopadhyay, Rahul Telang, Ray Reagans, Linda Argote, Alan Montgomery, Vijay Mookerjee, Debabrata Dey, Robert Jacobson, Eric Zivot, Corey Phelps, Ming Fan, Anjana Susarla, and seminar participants at Carnegie Mellon University, Purdue University, University of Washington, and Sun Yat-sen University for helpful suggestions. All errors are our own.
1. Introduction

In the last decade, the open source model has revolutionized traditional software development. The number of open source software (OSS) projects has grown dramatically in this time (von Hippel and von Krogh 2003, Weber 2004), and established software vendors such as IBM and Microsoft have begun to appropriate elements of open source in their own development efforts. As a result, academic research into various aspects of the OSS movement has exploded (von Krogh and von Hippel 2006). OSS is developed by voluntary developers working as a team. Central issues in the long term viability of this phenomenon deal with how to attract and sustain these developers in the absence of any contractual agreements (von Hippel and von Krogh 2003). Hence, a primary area of inquiry into OSS development focuses on how to direct, motivate and influence developer behavior (Markus et al 2000, Roberts et al 2006).

Though researchers have come up with several intrinsic and extrinsic motivations for developer participation in OSS, Roberts et al (2006) find that these motivations are not independent and in fact, related. An underlying motivation behind OSS developers’ participations is their desire to learn, i.e., to enhance their knowledge and skills through participation in OSS projects (Lakhani and Wolf 2005, Rossi and Bonaccorsi 2005). Learning literature states that learning creates a growing stock of knowledge and skills that can be applied in future to improve productivity (Argote and Epplle 1990, Darr et al 1995, Reagans et al 2005). Developers build long term capabilities through participation in OSS which can be transferred to new environments, resulting in increased job wages (Lerner and Tirole 2002, Hann et al 2002). Literature in psychology suggests that the knowledge and skills of an individual combine with her motivations to generate task specific behavior (Campbell and Prichard 1976). Hence, a potential way to influence developer behavior and motivate added contributions is by providing a richer environment for developer learning in an OSS project.

Although learning from own experience and from peers has been studied in a commercial software development environment (Boh et al 2007), prior research has not investigated learning in OSS environment. The effects of experience on developer productivity may differ between commercial and OSS development environments due to several reasons. First, in a commercial software development environment goals are clear, division of labor is pre-assigned, actions to follow are specified in advance,
and support processes are clearly defined. In contrast, OSS represents an ambiguous organizational setting where goals are often vague and shift over time, and code architecture evolves in an unstructured way resulting in meaning and utility of knowledge to vary across time and space, no pre-set paths to solve a problem exist and support processes are ill-defined (Raymond 1998, von Hippel and von Krogh 2003, Weber 2004). Second, there is significant diversity in the possessed knowledge and skills of developers attracted towards OSS and their joining times (Cox 1998, von Krogh et al 2003). Hence, OSS developers may differ in their abilities and preferences to learn from different activities (Cohen and Levinthal 1990, Shafer 2001). Third, whereas prior research does not account for the depreciation of learning in software development (Boh et al 2007), it is important to consider this in an OSS environment as OSS development is a part-time activity for most of the developers. An OSS developer may remain absent for extended periods during her involvement in the project. This may not only lead to depreciation of developer knowledge but also have an impact on the relevance of the possessed knowledge and skills as the code may evolve through contributions of peers during her absence. Fourth, enhancement of capabilities by learning may also influence performance indirectly by affecting an OSS developer’s intrinsic motivation to contribute (Roberts et al 2006), which are assumed to be held constant in a commercial environment through compensation schemes.

In this research, we propose a Hidden Markov Model (HMM) to capture developer learning dynamics. Our modeling framework allows us to investigate the influence of developer participation history in the project on her code contribution behavior. Two types of participations: past cumulative code contributions (own experience) and peer interactions are considered. We calibrate the model on six years of detailed data collected from 251 developers working on 25 OSS projects hosted at Sourceforge.

The proposed HMM has several advantages. First, the HMM allows segmentation of developers according to their code contribution and learning abilities. This segmentation allows investigating the varying impacts of learning activities across segments on subsequent learning and contributions. It provides insights into purposefully constructing structures and strategies so as to enhance or maximize developer learning and contributions in OSS environment, an immensely important but relatively unexplored area until now. Second, existing learning research suggests a static model of productivity,
which may lead to erroneous estimates of the impact of experience and interactions with peers if
dynamics in the code contribution behavior of developer exists. The HMM provides a simple way to
account for the possibility of change in task relevant behavior due to learning. Finally, the HMM provides
a way to control for developer specific unobserved heterogeneities in learning and code contributions, and
also accounts for serial dependency in the data.

The rest of the paper is organized as follows. In Section 2 we present the theoretical background. The
HMM to analyze developer learning is laid out in Section 3. Section 4 explains data collection
methodology. The variable description and estimation procedure are presented in Sections 5 and 6
respectively. The results from the HMM are discussed in Section 7. Section 8 offers concluding remarks.

2. Literature and Theory Development

In this section, we draw upon the vast prior literature to develop a dynamics learning theory for OSS
environment. We first determine the factors that affect developer productivity. Then, we build the theory
to explain why participation in learning activities (own experience and peer interactions) have the
potential to affect developer productivity. Finally, we discuss the challenges in modeling learning
dynamics and discuss how we address them.

2.1. Factors that Affect Developer Productivity

Rodney et al (1994) state that relevant variance in worker productivity in a specific task environment can
be explained by three factors (1) declarative knowledge (DK), (2) procedural knowledge and skills (PKS),
and (3) volitional choice (intrinsic motivation). DK represents knowing what to do, and is the knowledge
of facts, rules, principles and procedures (Kanfer 1990). PKS represent the abilities of knowing how and
being able to perform the task (Kanfer 1990). Intrinsic motivation is a combination of three behavioral
choices: (a) the choice to expend effort, (b) the choice of how much effort to expend and (c) actually
choose to work on the task with some effort for some period of time (Rodney et al 1994). According to
this framework, a situation (activity) can influence differences in worker productivity only by influencing
DK, PKS or intrinsic motivation. Learning curve literature investigates how experience or training affects
productivity through influencing DK and PKS (Argote et al 1990). Personnel economics literature
investigates how different compensation schemes (external incentives) affect a worker’s productivity through influencing her intrinsic motivations (Milgrom 1988). Studies in Industrial psychology suggest that enhancement of capabilities through past experience may affect the intrinsic motivations and hence productivity of an individual (Deci 1990).

2.1.1. Declarative and Procedural Knowledge and Skills in OSS

Software development activities, though not entirely repetitive, require a considerable amount of abstract, technical, theoretical and experiential knowledge (Boh et al 2007, Sacks 1994). Prior research suggests that the processes that lead to creation of new knowledge, embodied in complex artifacts such as software, most often involve a recombination of known conceptual and physical materials (Fleming 2001, Baldwin and Clark 2006, Narduzzo and Rossi 2003). Critical inputs into these processes are understanding and knowledge of current design problems, existing related solutions, novel approaches to solve these problems, and failed approaches; and related technical skills in areas such as relationships among data-items, algorithms, invocation of functions, code architecture, among others (Fleming 2001, Grewal et al 2006, Sacks 1994, Singh 2007, Singh et al 2007). Through involvement in software development activities a developer may build her own knowledge repositories which can be applied in related situations in future (Basili and Caldiera 1995, Sacks 1994). However, some knowledge may also become irrelevant due to new advancements or significant contributions to the code by peers. It may also depreciate due to forgetting as a result of inactivity on the part of a developer.

2.1.2 Volitional Choice or Intrinsic Motivation in OSS

Deci (1975) states that if an individual’s feelings of competence and self-determination are enhanced (diminished), her intrinsic motivation will increase (decrease). Hence, an individual’s past experiences in the project that positively affect her perceived competence and self determination would positively affect her productivity. An individual’s perceived feelings of internal control, self efficacy and levels of enjoyment from past tasks would influence the amount of effort she would be willing to exert (Leper and Henderlong 2000, Benabou and Tirole 2002). Roberts et al (2006) find that higher levels of intrinsic motivations reduce the sensitivity to extrinsic incentives for OSS developers.
2.2. Effect of Learning Activities on Developer Productivity

Developers are endowed with certain amount of DK and PKS when they join an OSS project. Extant research has also found that OSS developers differ in their intrinsic motivations to contribute (Lakhani and Wolf 2005, Roberts et al 2006). Recent research has suggested that the initial endowment and intrinsic motivations of an individual may be attenuated or accentuated by subsequent experiences with same or related processes in determining future outcomes (productivity) of a dynamic process (Heckman 1991). The absorptive capacity literature states that the possessed DK and PKS of an individual may affect her ability to identify and assimilate knowledge from the environment which further affects her productivity (Cohen and Levinthal 1990).

2.2.1 Own Experience

Own experience primarily affects the procedural knowledge and skills of a developer. Expertise in programming is known to produce an order of magnitude improvement in program efficiency (Brooks 1987). In any innovative process such as software development, technical problems are solved through an adaptive process (Sacks 1994). Such an adaptive process involves a person undertaking a course of action, the environment producing a result, and the person updating her course of action to increase the propensity of achieving her goals (March and Olsen 1972, Van de Ven and Polley 1992). By solving technical problems or developing code, a developer acquires knowledge of not only what works in a given situation but also what does not work in such a situation. The more a developer participates in such activities, the richer her repositories of such knowledge which can be applied in future. Own experience may also affect declarative knowledge of a developer by her introduction to datasets, relationships among data-items, algorithms, invocation of functions, code architecture, among others (Brooks 1987, Weber 2004). Developers acquire competence as their knowledge and skills increase through their past involvements in the project (Campbell and Prichard 1976). Past involvement results in enhancements in perceived control and self esteem due to peer recognition and can raise a developer’s level of enjoymen in the task making her more intrinsically motivated to contribute (Roberts et al 2006). Own experience may thus enhance the DK, PKS and intrinsic motivations of a developer.
2.2.2. Peer Interactions

Extant research on learning from others focus on the role of transfer mechanism, conduits or agents through which transfer of knowledge takes place, in facilitating the knowledge transfer process (Darr et al. 1995). In general, the literature suggests that higher levels of use of transfer mechanism are associated with increased levels of knowledge transfers. The transfer mechanism particularly relevant in the present study is interactions with peers. Psychology research posits that at the group level information may be distributed across individuals, and individual members may draw upon social cognition to solve problems (Larson and Christensen 1993). Brown and Duguid (1991) argue that expertise is context dependent and evolves through interactions. Interactions with peers are an important source of learning for developers in an OSS project due to several reasons. First, these interactions allow opportunities for resource pooling, knowledge spillovers and sharing alternative interpretations of the design problems. A developer can draw upon the knowledge repositories of peers in solving a problem. Second, these interactions help in coordinating the development effort which may minimize effort duplication besides updating the community about recent advancements to the code (Faraj and Sproull 2000, Kraut and Streeter 1995). Third, it provides opportunities to a developer for applying her efforts or knowledge to different but related problem domains and in the process developing a deeper cognitive understanding of both (Schilling et al. 2003).

Lakhani and Hippel (2003) find that 98% of the effort expended by information providers in fact returns direct learning benefits to those providers on Apache field support systems. Peer interactions provide developers the opportunity to help others solve their problems resulting in increased feeling of satisfaction and hence higher intrinsic motivation to participate. Through participation in peer interactions a developer may learn and get intrinsically motivated to offer something back to the community thus affecting her contributions (Lakhani and Hippel 2003). Taken together, participation in learning activities (own experience and peer interactions) have the potential to impact developer productivity through affecting her DK, PKS or intrinsic motivations.
2.3. Challenges in Modeling Learning Dynamics

2.3.1. Latency of DK, PKS and Intrinsic Motivation

The major challenge in modeling learning dynamics is the latency of DK, PKS and intrinsic motivations of a developer. We follow the existing learning curve literature and do not aim to determine the exact impact of past performance on DK, PKS and intrinsic motivations (Argote et al 1990, Boh et al 2007). We determine how participation in learning activities in past affect future productivity.

In the econometric modeling literature, the impact of past behavior on subsequent outcomes of interest is called state dependence (Heckman 1991). The learning curve literature models the state dependence in worker productivity through the incorporation of own experience in the production equation (Darr et al 1995). The underlying reason for state dependence in these studies is that experience creates a growing stock of latent factors such as DK and PKS (Argote et al 1990). Worker intrinsic motivations in a commercial environment are assumed to be constant through fixed extrinsic incentives in learning curve studies. However, the experience may also capture the change in intrinsic motivations if any, as these studies do not examine the specific mechanism through which experience affects productivity.

2.3.2. Dynamics in Learning and Code Contributions Behavior

A significant limitation of the learning curve literature is its restrictive account of worker behavior dynamics. Though it includes a state dependence term the rest of the model is static. If there are worker behavior dynamics (as would be the case if enhancement of knowledge and skills make a developer less/more sensitive to extrinsic incentives by affecting her intrinsic motivations) then the learning curve model may produce erroneous estimates.

2.4. Hidden Markov Model for Developer Learning in OSS

Based on the above discussions, we argue that a comprehensive model of developer learning in OSS should (1) be able to capture the change in developer code contribution behavior as a result of participation in learning activities, (2) account for the relevant knowledge and the skills set of a developer to influence her ability to identify, assimilate and exploit knowledge from the environment, and (3) allow for depreciation in DK, PKS and intrinsic motivations.
To operationalize these aspects of learning, we propose a HMM,\(^1\) similar to the state dependent models, which exploits the individual participation history in order to analyze the inter-temporal variance in developer code contributions. The Markovian states of HMM map to a finite set of hidden states. For lack of a more appropriate term, we call this state a learning state. The space of a state is spanned by multiple (latent) factors (DK, PKS, and intrinsic motivation) whose values can be varied continuously whenever appropriate. Each learning state represents a specific contribution behavior of a developer. While we acknowledge that knowing does not always transform into doing, a useful conception of learning must include change, such that an individual is said to learn if her actions change as a result of reflection on new knowledge or insights.

A developer can transition from one state to another through participation in observed learning activities over a given period. The Markovian transitions account for the dependence of subsequent learning on the present state of a developer (the state transition in period \(t\) depends only on the state of the developer in period \(t - 1\)). The code contributions of the developer are modeled as an observed stochastic process. The learning is structurally modeled through an integrated framework that links the unobserved but evolving latent state of learning with the realized outcomes of learning – developer code contributions (see Figure 1 for a graphical representation of the proposed HMM). Through this model, we investigate the impact of learning activities that affect the transitions of a developer between the hidden states.

\[\text{Figure 1: Hidden Markov Model of Developer Learning in OSS}\]

Our model is structurally different from the models in the existing learning curve literature. There are several advantages of our model. First, the state dependent models are static models of individual

---

\(^1\) A Hidden Markov Model (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).
production behavior. The HMM accounts for the possibility of code contribution behavior dynamics of a developer. Second, the model allows to probabilistically identifying the learning state of a developer and investigating the impact of learning activities on moving the developer to a different state.

3. Modeling Developer Learning Dynamics

Consider an OSS project. The entire time horizon starting from the inception of the project is divided into periods. For each period, we observe the factors that may affect the code contributions of the developer as well as her actual code contributions, if any. For any given period, a developer resides in an unknown learning state.

Given a set of learning states \( s \in \{1, 2, \ldots, n\} \), where 1 is the lowest learning state and \( n \) the highest, and a sequence of observed contributions \( O = O_1 O_2 \cdots O_T \) (\( O_i \in \{\text{amount of code contributed}\} \) in our model), the HMM is comprised of three elements: (i) the initial state distribution, \( \pi \); (ii) the state-transition probability distribution, \( Q \); and (iii) the observed outcome probability vector, \( A \). A HMM requires the specification of \( n \) (number of states) and the three probability measures \((\pi, Q, A)\). For convenience, we use the following compact notation, \( \lambda = (\pi, Q, A) \), to represent the complete parameter set of the model.

Consider, for developer \( i \), a fixed state sequence \( S(i) = S_{i1} S_{i2} \cdots S_{iT} \), where \( S_{i1} \) is the initial state for developer \( i \) and \( S_{iu} \in \{1, 2, \ldots, n\} \), and an observed outcome sequence \( O(i) = O_{i1} O_{i2} \cdots O_{iT} \). The probability of the observed outcome sequence, \( O(i) \), for the state sequence \( S(i) \) and the parameter set \( \lambda \) is given by

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

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

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

where \( a(O_{it} | S_{it}) \) is the probability of observing contribution \( O_{it} \) given that developer \( i \) is in state \( S_{it} \) at time \( t \). Note that \( a(O_{it} | S_{it}) \) is an element of the contribution probability vector, \( A(i, t) \). The probability of such a state sequence \( S(i) \) is given as:
where \( \pi(i) \) is the initial probability that developer \( i \) is in state \( S_{i1} \) in period \( t = 1 \). \( q(S_i, S_{i+1}) \) is the probability that developer \( i \) is in state \( S_{i+1} \) in period \( t + 1 \) given she was in state \( S_i \) in period \( t \). Note that \( q(S_i, S_{i+1}) \) 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

\[
P(O(i), S(i) | \lambda) = P(O(i) | \lambda, S(i)) P(S(i) | \lambda).
\]

Hence, the probability of the observed outcome sequence \( O(i) \) given the model \( \lambda \) is also the likelihood of observing this sequence and is obtained by summing Equation (1) over all possible values of state sequence \( S(i) \) (Rabiner 1989), explicitly,

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

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

\[
L(O(i)) = \pi(i) \Lambda(i, 1) Q(i, 1, 2) \Lambda(i, 2) Q(i, 2, 3) \cdots Q(i, T - 1, T) \Lambda(i, T) \mathbf{1}^T.
\]

Here, the matrix \( \Lambda(i, t) = \text{diag}(a(O_i | S_{i1} = 1), a(O_i | S_{i2} = 2), \ldots, a(O_i | S_{in} = n)) \) and \( \mathbf{1} \) is a \( n \times 1 \) vector of ones. To obtain the likelihood of the observed outcome sequence \( L(O(i)) \), we need the model parameters \( Q, A, \) and \( \pi \). These parameters are defined in the following subsections.

### 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 non-zero probabilities to transitions to non-adjacent states. We allow transitions to lower states to account for the temporal nature of the relevancy of possessed knowledge and skills as well as possible lessening in intrinsic motivation. The random walk state transition matrix is defined as follows:
\[ Q(i, t, t+1) = \begin{pmatrix}
q(1,1) & q(1,2) & 0 & \cdots & 0 & 0 \\
q(2,1) & q(2,2) & q(2,3) & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & q(n,n-1) & q(n,n)
\end{pmatrix}. \]

Here, \( q(j,k) = q(S_u = j, S_{u+1} = k) = P(S_{u+1} = k | S_u = j) \), and for each state \( j \), \( \sum_{k=1}^{n} q(j,k) = 1 \) and \( 0 \leq q(j,k) \leq 1 \ \forall \ j, k \in \{1,2,\ldots,n\} \).

This probabilistic transition is modeled by considering a propensity to transition which is affected by a developer’s participations in learning activities. 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, \( q(S_u, S_{u+1}) \) is modeled as an ordered logit:

\[
q(j, j+1) = 1 - \frac{\exp(\mu(h, j) - \beta_j R_u - \xi_j)}{1 + \exp(\mu(h, j) - \beta_j R_u - \xi_j)}, \quad q(j, j-1) = \frac{\exp(\mu(l, j) - \beta_j R_u - \xi_j)}{1 + \exp(\mu(l, j) - \beta_j R_u - \xi_j)}, \quad \text{and}
\]

\[
q(j, j) = \frac{\exp(\mu(h, j) - \beta_j R_u - \xi_j)}{1 + \exp(\mu(h, j) - \beta_j R_u - \xi_j)} - \frac{\exp(\mu(l, j) - \beta_j R_u - \xi_j)}{1 + \exp(\mu(l, j) - \beta_j R_u - \xi_j)}, \quad \forall j \in \{1,2,\ldots,n\}. \tag{3}
\]

Here, \( \mu(h,n) = \infty \), \( \mu(l,1) = -\infty \), and \( R_u \) is a vector of variables that represent participation in learning activities by developer \( i \) in period \( t \). The variables included in the vector \( R_u \) are explained in Section 5.1. \( \beta_j \) is a parameter vector for impact of developer learning activities, \( R_u \), on the propensity to transition from state \( j \). The developer specific unobserved heterogeneity for state transitions is captured by the developer specific random effects, \( \xi_j \). The threshold value to move to an upper state is represented by \( \mu(h, j) \), whereas the threshold value to move to a lower state is represented by \( \mu(l, j) \). The constraint \( \mu(h, j) > \mu(l, j) \) is applied to ensure that the high threshold is larger than the low threshold. Notice that in an intermediary state \( j \in \{2,3,\ldots,n-1\} \), 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, 1, a developer can either
move up one state or stay in the same state. Similarly, in the highest state, \( n \), a developer can either move down one state or stay in the same state.

### 3.2. State Dependent Code Contribution Probability

Extant software development research suggest the use of completions of Modification Request (MR) as a measure of productivity for projects that follow incremental software development approach (Boh et al 2007). MR’s are similar in concept to work orders and are used to add new functions and modify or repair old functions (Ibid). In OSS, the Concurrent Versioning System (CVS) commit transaction\(^2\) represents a basic change similar to the MR in commercial development environment (Mockus et al 2000). Hence, we use number of CVS commits by a developer as a measure of her productivity. Since number of CVS commits is a count measure, traditional approach would be to model it as a Poisson process. However, the Poisson process contains the strong assumption that the mean equals variance. In our data, the variance exceeds the mean which causes over-dispersion. Though, in this scenario, a Poisson model would estimate the parameters consistently, the standard errors would be underestimated. However, a Negative Binomial estimation which is a generalization of the Poisson model allows for over dispersion by relaxing the mean equal to variance assumption (Hausman et al 1984). Hence, we employ the Negative Binomial distribution to model the state dependent contribution probabilities. The Negative Binomial Model is:

\[
Pr\left(\text{Number of CVS Commits} = O_i \right) = \frac{\lambda_j^{O_i} \cdot O_i^{O_i} \cdot e^{-\lambda_j} \cdot \Gamma(O_i + 1)}{\Gamma(O_i) \cdot \Gamma(1)},
\]

where \( O_i \) is the number of CVS commits made by developer \( i \) in period \( t \); \( \lambda_j = \exp(\beta \cdot W_i + \eta_i + \epsilon_{ij}) \) is the conditional mean; \( W_i \) is the vector of inputs into a developer’s coding process; \( \beta \) is a vector of state dependent parameters; \( \epsilon_{ij} \) is the error term which follows a log-gamma distribution (Greene 2007). Note that in a Poisson model this term is zero. \( \eta_i \) is the developer specific random effects that capture the developer specific unobserved heterogeneity.

The probability of observing a given number of CVS commits is then given as
\[
a(O_n | S_n = j) = \frac{\Gamma(\theta_j + O_n)}{\Gamma(O_n + 1) \Gamma(\theta_j)} h_{\theta_j}^\theta_j (1 - h_{\theta_j})^{\theta_j}, \quad \text{where} \quad h_{\theta_j} = \frac{\exp(\rho W_n + \eta_i)}{\exp(\rho W_{\theta_j} + \eta_i) + \theta_j}.
\]

Here, \(\theta_j\) and \(\rho_j\) are the parameters that need to be estimated. The variables that constitute \(W\) are explained in Section 5.2. The likelihood for developer \(i\) is then given as

\[
L(O(i)) = \int \int L(O(i) | \eta, \xi) dG(\eta | \xi) dH(\xi).
\]  (4)

Note that we allow the individual specific random effects for learning \((\xi)\) and coding \((\eta)\) to be correlated.

4. Data

Data was collected from the projects hosted at Sourceforge.net (SF) (Madey 2006). SF is the world’s largest OSS project repository and accounts for approximately 90% of all OSS projects. It provides a good sample of the OSS community to study the underlying dynamics. We considered only those projects that were incepted at Sourceforge during the six month period from January 1, 2000 to June 30, 2000. We further considered only those projects that satisfied three criterions (1) project must use CVS at Sourceforge, (2) all associated developers of the project should be authorized to commit to CVS, and (3) project’s email list at Sourceforge should be its only mentioned source of developer discussions. The first condition is required to ensure that the developer code contribution data is available. In some projects only a selected few are given the power to commit to the CVS. This hampers the tracking of development activity. The second condition ensures that developer activity is correctly tracked. Some of the projects use other platforms for discussion rather than the mailing lists at Sourceforge. Due to 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 or use IRC channels for discussion.

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 provided us

---

\(^2\) Software developers use CVS to manage software development process. CVS stores the current version(s) of the project and its history. A developer can check-out the complete copy of the code, work on this copy and then check in her changes. The modifications are peer reviewed. The CVS updates the peer reviewed modified file.
approximately 6 years of data for 25 projects. The CVS commit and developer mailing list data for developers who appear at least 10 times in the dataset 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.

<table>
<thead>
<tr>
<th>Table 1: Mean Statistics of Data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overall Statistics</strong></td>
</tr>
<tr>
<td>Number of Projects</td>
</tr>
<tr>
<td>Number of Developers</td>
</tr>
<tr>
<td>Number of Coding Opportunities</td>
</tr>
<tr>
<td>Mean Number of Periods per Developer</td>
</tr>
<tr>
<td>Length of a Period</td>
</tr>
<tr>
<td><strong>Project Level Statistics</strong></td>
</tr>
<tr>
<td>Mean (Sourceforge) Rank of Projects</td>
</tr>
<tr>
<td>Mean Number of Registered Developers Per Project</td>
</tr>
<tr>
<td>Mean Number of Developers Considered Per Project</td>
</tr>
<tr>
<td><strong>CVS Statistics</strong></td>
</tr>
<tr>
<td>Mean Number of Periods of Commits Per Developer</td>
</tr>
<tr>
<td>Mean Number of Total Lines Added Per Developer</td>
</tr>
<tr>
<td><strong>Developer Mailing List Statistics</strong></td>
</tr>
<tr>
<td>Mean Number of Threads Started by a Developer</td>
</tr>
<tr>
<td>Mean Number of Threads Participation by a Developer</td>
</tr>
<tr>
<td>Mean Number of Threads Per Project</td>
</tr>
<tr>
<td>Mean Number of Days a Thread spans</td>
</tr>
</tbody>
</table>

The data about code contributions of developers in a project is extracted through the CVS log entries.³ The peer interactions data was collected from the archives of the developer mailing lists⁴ 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 human effort. The email information we use is based on threaded correspondences because that ensures that the original

³ Note that we do not consider bug fixing as a separate contributing activity in this model as any resulting modification in the source code is recorded in the CVS log files.

⁴ The mailing lists were combined for projects that had more than one mailing list for developers.
post was not just an announcement and demanded a reply. Project related data was collected from the project homepages at Sourceforge.net. Mean statistics of the data collected are presented in Table 1.

5. Variable Descriptions

In this section, we describe the variables that constitute $W_u$ and $R_u$.

5.1. Variables Impacting Learning

These variables constitute the vector $R_u$ used to calculate the state-transition probabilities. These variables represent either of the two modes of learning for an OSS developer. These are recorded after the first observation of a developer in the project. First observation of a developer is the first time she appears in the mailing list or the CVS commit logs.

Learning curve studies have modeled learning as a function of cumulative experience of performing a task (Argote et al 1990, Boh et al 2007). In our model, the learning by own experience mode is represented by the amount of developed code accumulated to the previous time period. Learning from peers is captured through a developer’s involvement in threaded communication. We capture two different aspects of a developer’s involvement in threaded communication. A thread usually serves the interests of a developer who started it by resolving her problem or through discussion of issues in which she may be interested. To capture this effect, we consider the number of threads started by a developer in the current period. 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 herself. This might serve the interests of a developer directly or indirectly. To capture this effect, we consider the number of threads that a developer did not start but in which she participated for the current period. The variables that constitute vector $R_u$ are operationalized as described in Table 2.

5.2. Variables Impacting Code Contribution

Table 2 presents the variables that influence the code contribution of a developer in each time period. They constitute the vector $W_u$ used to calculate state dependent code contribution probabilities. These are also recorded after the first observation of a developer in the project. We categorize these variables as: developer characteristics, project characteristics and the project life cycle effects.
### Table 2: Model Variables

<table>
<thead>
<tr>
<th>Learning Activities ($R_{it}$)</th>
<th>Code Contribution ($W_{it}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines Committed $it_{-1}$</td>
<td>Manager</td>
</tr>
<tr>
<td>Cumulative number of source code lines committed by developer $i$ till period $t - 1$</td>
<td>Dummy variable equals 1 if the developer is manager of the project and 0 otherwise</td>
</tr>
<tr>
<td>Threads Started $it$</td>
<td>Project Rank $i_{t-1}$</td>
</tr>
<tr>
<td>Number of threads started by developer $i$ in period $t$</td>
<td>Sourceforge Project Rank in period $t - 1$. 1 is the highest rank.</td>
</tr>
<tr>
<td>Threads Participated $it$</td>
<td>Involvement Quotient $i_{t}$</td>
</tr>
<tr>
<td>Number of threads in which developer $i$ participated but did not start in period $t$</td>
<td>Number of months the developer showed activity since joining the project</td>
</tr>
</tbody>
</table>

**Developer Characteristics.** A developer’s code contributions to an OSS project are determined by her DK, PKS, intrinsic motivation, the presence (absence) of external situations that affect intrinsic motivations and availability. The learning state specific constant term in the contribution equation captures the differences in code contributions of developers owing to differences DK, PKS and intrinsic motivations. Presence of extrinsic incentives such as status and future economic incentives may affect a developer’s intrinsic motivations to contribute (Roberts et al 2006). Such extrinsic incentives are closely tied with a developer and a project’s visibility (Lerner and Tirole 2002 and Roberts et al 2006). Project managers receive greater visibility just owing to their higher status (Hann et al 2005). We control for this effect through computation of a variable ‘Manager’ representing whether a developer is a manager on the project or not. A project’s visibility is tied to its performance. Sourceforge provides a composite measure of project performance (‘Project Rank’) by ranking the projects at the end of each month based on its activity and popularity among developers and users (Crowston et al. 2003). Besides extrinsic incentives, long time association with a project may make the developer more loyal to the project and, hence, may
influence her contributions. We account for such loyalty by controlling for a developer’s level of past involvement in the project (‘Involvement Quotient’). Since participation in OSS is not a full time activity for most of the developers, they may not be available for code contributions in each period. We control for this effect by incorporating a variable ‘Availability’ in period $t$ as a dummy which equals to 1 if the developer is involved in the project in period $t$ or 0 otherwise. Developer’s motivations to participate in OSS such as creative pleasure, altruism or intrinsic desire to fight against proprietary software may affect her contributions but are difficult to observe and hence are controlled through the use of developer specific unobserved effects.

**Project Characteristics.** We control for those characteristics of projects that may influence quality of developers attracted and the quality bars for accepting code contributions. We control for whether the project is a software development related project or not. We also control for whether the intended audience is technical (i.e. system administrators/developers) or not. We specifically control for these type and intended audience as projects with these characteristics are likely to attract sophisticated developers. Hence, these projects would provide an ideal environment for such developers who want to accrue reputation benefits to contribute and show their programming prowess to sophisticated peers. As CVS commits are peer reviewed, these projects may have high bar for accepting contributions.

**Project Life Cycle Effects.** Project life cycle effects such as code complexity or code maturity with time may affect code contributions. To control for the effect of project life cycle on code contribution we calculate a variable ‘Project Age’. To account for any non linearity in the effect of Project Age on code contributions, we also include variable ‘Project Age Squared’ in the model.

6. Estimation Procedure and Model Selection Criteria

6.1. Some Econometric Issues

There are three main theoretical and statistical challenges that may lead to inconsistent estimates: unobserved heterogeneity, reverse causality and serial correlation (Greene 2007). In this study, unobserved heterogeneity may exist at two levels – learning state specific and developer specific. Further, at each level the unobserved heterogeneity may exist for learning behavior or code contribution behavior. For instance, owing to more knowledge and skills, developers in high state could be more likely to be
available more often or show more involvement or participate more in peer interactions. The HMM controls for these state specific unobserved effects through the inclusion of state specific constant terms in the contribution equations and state specific threshold parameters in transition equations. This specification is similar to state specific fixed effects estimation. In the present study, developer specific unobserved heterogeneity refers to the possibility that unmeasured (or immeasurable) differences (such as intrinsic motivation or external learning opportunities available to a developer) among observationally equivalent developers may affect their learning as well as code contributions. This unobserved heterogeneity is controlled for in two ways in the model. The time varying changes in the knowledge stocks/abilities of the developers are being captured by the learning states. The developer specific inherent time in-varying unobserved effects are controlled for through their explicit modeling as developer specific random effects $\eta$ and $\xi$ (Altman 2007). The problem of reverse causality is minimal in the present model as the participation in learning activities are lagged with respect to the code contributions, i.e. participation in learning activities in period $t$ only affect code contributions in period $t + 1$. Finally, the state-space structure of our model accounts for the serial correlation, leading to consistent and efficient estimates (MacDonald and Zucchini 1997, Roweis and Ghahramani 1999).

6.2. Estimation Procedure

Maximum Likelihood Estimation is used for parameter estimation from Equation (4). Equation (2) is evaluated at a give value of $\eta$ and $\xi$ to get the value of $L(O(i) | \eta, \xi)$ which is inserted into Equation (4). To avoid any misspecification of heterogeneity distributions $G$ and $H$, we estimate them non-parametrically (Heckman and Singer 1984). This involves approximating the underlying unknown probability distribution by a finite number of support points for $\eta$ and $\xi$, and the location and probability mass associated with them. We relate two normalizing constants with the support points, and set the bounds on each of the random effects to be 0 and 1. We follow an iterative procedure and add support points until the inclusion of an additional point leads to a situation where two support points overlap. We used sequential BFGS Newton-Raphson algorithm to maximize the likelihood given in Equation (4) (LeSage 2005). As is common in this approach, we ensured the stability of the results by running the
analysis with several different randomly selected starting values for the parameters as well as initial state distributions.

One issue that remains to be considered is how to choose the number of states $n$. Greene and Hensher (2003) suggest the use of Bayesian Information Criterion (BIC) for model selection:

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

Here $\text{size}$ is the number of parameters in the model, and $D$ is the number of developers. The models are run for increasing number of states starting with one state and going up to four states. These scenarios are run separately and their log-likelihood values are obtained. The log-likelihoods for all scenarios are shown in Table 3. The three-state HMM outperforms all other model specifications. The one-state model assumes static code contribution behavior for a developer. However, it is outperformed by all the other models that assume dynamic code contribution behavior. This implies that own experience and peer interactions cause a change in code contribution behavior of a developer.

<table>
<thead>
<tr>
<th>Table 3: Comparison of HMM Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>One</td>
</tr>
<tr>
<td>Two</td>
</tr>
<tr>
<td>Three</td>
</tr>
<tr>
<td>Four</td>
</tr>
</tbody>
</table>

7. Results, Discussion and Limitations

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

7.1. Learning State – Code Contribution Behavior Relationship

The results indicate that a change in learning states causes a change in code contribution behavior of the developers. For example, a transition from state 2 to state 3, makes a developer’s coding decision insensitive to prior period’s project performance, besides increasing her capabilities to code captured through the state dependent constant term. Since lower values of project rank indicate better performance for the project, the developers in states 1 and 2 are more likely to contribute when the project is doing well.
Table 4: Estimated Parameters for the Three State HMM

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Learning States</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>State 1</td>
</tr>
<tr>
<td>State Transition ($\beta$)</td>
<td></td>
</tr>
<tr>
<td>Variable Type</td>
<td>Variable Name</td>
</tr>
<tr>
<td>Learning from Own Experience</td>
<td>Lines Committed $t-1$</td>
</tr>
<tr>
<td>Learning from Peers</td>
<td>Threads Started $t$</td>
</tr>
<tr>
<td> </td>
<td>Threads Participated $t$</td>
</tr>
<tr>
<td>Upper Threshold</td>
<td>$\mu_{h,s}$</td>
</tr>
<tr>
<td>Lower Threshold</td>
<td>$\mu_{l,s}$</td>
</tr>
<tr>
<td>Coding ($\rho$)</td>
<td></td>
</tr>
<tr>
<td>Variable Type</td>
<td>Variable Name</td>
</tr>
<tr>
<td>Capability</td>
<td>Constant</td>
</tr>
<tr>
<td>Extrinsic Incentives</td>
<td>Manager</td>
</tr>
<tr>
<td> </td>
<td>Project Rank $t-1$</td>
</tr>
<tr>
<td>Loyalty</td>
<td>Involvement Quotient $t$</td>
</tr>
<tr>
<td>Availability</td>
<td>Availability $t$</td>
</tr>
<tr>
<td>Project Characteristics</td>
<td>Software Development</td>
</tr>
<tr>
<td> </td>
<td>Technical Audience</td>
</tr>
<tr>
<td>Project Life Cycle Effects</td>
<td>Project Age $t$</td>
</tr>
<tr>
<td> </td>
<td>Project Age Squared $t$</td>
</tr>
<tr>
<td>Dispersion ($\theta$)</td>
<td>1.473***</td>
</tr>
<tr>
<td>Unobserved Heterogeneity ($\eta$, $\xi$)</td>
<td></td>
</tr>
<tr>
<td>$C_{\eta} = −0.190$, $C_{\xi} = −0.121$</td>
<td></td>
</tr>
<tr>
<td>Probability</td>
<td>0.027</td>
</tr>
<tr>
<td>Conditional Distribution: Probability($\xi$</td>
<td>$\eta$)</td>
</tr>
<tr>
<td>$\eta_1 = 0$</td>
<td>0.9915</td>
</tr>
<tr>
<td>$\eta_2 = 0.178$</td>
<td>0.0029</td>
</tr>
<tr>
<td>$\eta_3 = 1.000$</td>
<td>0.0055</td>
</tr>
</tbody>
</table>

*** $p < 0.01$, ** $p < 0.05$, * $p < 0.1$ (two tailed $t$-test for all variables)

The variables were checked for problems of multicolinearity. The following scaling was performed to ensure solution stability and reduce correlation between variables: Project age variable is calculated as the (True Project Age −36)/100. Project Age Squared variable is calculated as (True Project Age −36)$^2$/10000. This transformation is performed to reduce the correlation between the two age related variables (Gelman and Hill 2007). Threads Started, and Threads Participated, are scaled down by a factor of 10. Involvement Quotient $\alpha$, Lines Committed $t-1$ and Project Rank $t-1$ are scaled down by a factor of 100, 10000 and 100,000 respectively.
More knowledgeable and skilled developer can direct the growth and architecture of the code by making major contributions. Hence, as developer gains knowledge and skills, she would become less sensitive to a project’s performance as she has the potential to significantly influence future performance. For the projects that are software development related or aimed at technical audience, developers are more likely to contribute when they are in the states 2 and 3 and are less likely to contribute when they are in state 1. This confirms our belief that it may be tougher to contribute to such projects. Manager contributes more than developers in each of the three states. Developers in all three states are less likely to contribute code as the project grows older. As a developer’s active involvement in the project grows, so does her contributions across the three states. Note that on average a developer is likely to contribute most amount of code when in state 3 and least amount of code when in state 1. For example, consider a software development project aimed at technical audience. At the mean levels of all other variables that go into W the mean number of CVS commits by developer in such a project in state 1, state 2 and state 3 are 0.002, 3.337, and 42.444 respectively.⁶ Note that there will be huge variation in the number of CVS commits around these mean values as indicated by high values of state specific dispersion parameters in Table 4. For the ensuing analysis we refer to states 1, 2 and 3 as low, medium and high respectively.

7.2. Learning 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. The probabilities in the transition matrices are calculated from Equation (3) and a sample calculation is shown in the Online Supplement to this paper.

<table>
<thead>
<tr>
<th></th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>t→t+1</td>
<td>95.60%</td>
<td>4.40%</td>
<td>0%</td>
</tr>
<tr>
<td>Low</td>
<td>12.58%</td>
<td>84.45%</td>
<td>2.97%</td>
</tr>
<tr>
<td>Medium</td>
<td>0%</td>
<td>16.06%</td>
<td>83.94%</td>
</tr>
</tbody>
</table>

Table 5 shows that all the states are extremely ‘sticky’, i.e. developers are likely to stay in the same state if they stay idle for one period. One implication of this finding is that once a developer moves up in

⁶ See Online Supplement to the paper for these calculations.
learning states she is more likely to stay there. This signifies the persistence of learning, and, hence, a persistent change in code contribution behavior. Also note that the probability to transition to a lower state is much higher than the probability to transition to a higher state. These non-zero intrinsic probabilities to transition to adjacent states illustrate the temporal nature of the utility and relevance of possessed knowledge in OSS development.

7.3. Learning from Peers

Developers in all the three learning states benefit from interactions with peers. The associated coefficients are positive and significant for all the three states. Table 6 indicates the impact of participation in interactions on the learning of a developer with average amount of Thread Starts and Thread Participated. Asking questions by starting threads helps a developer in the medium state, the most. This increases the probability of transition to the high state from 2.97% to 6.49%, besides reducing her probability to transition to low state from 12.58% to 5.97%. A developer in the high state also increases her likelihood to stay in the same state from 83.94% to 87.61%. These changes are highly significant given the stickiness of the states and amount of code contributions in the higher state. However, a developer in the low learning state does not learn much by starting threads. This may be due to the reason that a developer in the low learning state may not be able to frame her questions properly, resulting in other developers either not understanding the question or finding it difficult to answer (Raymond and Moen 2001). Besides, it also indicates that the developer in the low state may lack in possessed relevant knowledge that may be required to comprehend other’s responses to her questions.

<table>
<thead>
<tr>
<th></th>
<th>Threads Started</th>
<th>Threads Participated</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t \rightarrow t + 1$</td>
<td>$t \rightarrow t + 1$</td>
</tr>
<tr>
<td>Low</td>
<td>95.14%</td>
<td>Low</td>
</tr>
<tr>
<td>Medium</td>
<td>5.97%</td>
<td>Medium</td>
</tr>
<tr>
<td>High</td>
<td>0%</td>
<td>High</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>95.14%</td>
<td>4.86%</td>
<td>0%</td>
</tr>
<tr>
<td>Medium</td>
<td>5.97%</td>
<td>87.54%</td>
<td>6.49%</td>
</tr>
<tr>
<td>High</td>
<td>0%</td>
<td>12.39%</td>
<td>87.61%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>85.19%</td>
<td>14.81%</td>
<td>0%</td>
</tr>
<tr>
<td>Medium</td>
<td>9.15%</td>
<td>86.66%</td>
<td>4.19%</td>
</tr>
<tr>
<td>High</td>
<td>0%</td>
<td>10.69%</td>
<td>89.31%</td>
</tr>
</tbody>
</table>

Participation in discussions started by others has a huge impact on the learning of developers in the low and the high state. A developer in the high state may benefit as good questions help developers gain their understanding, and often reveal problems they might not have noticed or thought about otherwise. It
increases her probability to stay in the high state from 83.94% to 86.66%. As Table 6 shows, the probability of transition for a developer in the low state to move to the medium state increases from 4.40% to 14.81%. This increase is quite significant given the “stickiness” of the low state.

7.4. Learning from Own Experience

The coefficients for Lines Committed are positive and significant for developers in the medium and high learning states but insignificant for a developer in the low learning state. A developer in the low state may not possess the knowledge relevant to the project and hence depend upon peers to acquire that knowledge. Though Learning from own experience has a positive impact on the learning of a developer in medium or high state, the effects are much less pronounced. This indicates that developers who contribute significantly are experienced in coding before joining the project. However, the positive impact of coding may be just due to the learning of the architecture and design of the source code specific to the project. Table 7 shows the transition matrix for the impact of code development on the learning of a developer. 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 \( t \).

<table>
<thead>
<tr>
<th>( t \rightarrow t + 1 )</th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>95.60%</td>
<td>4.40%</td>
<td>0%</td>
</tr>
<tr>
<td>Medium</td>
<td>12.37%</td>
<td>84.60%</td>
<td>3.03%</td>
</tr>
<tr>
<td>High</td>
<td>0%</td>
<td>15.67%</td>
<td>84.33%</td>
</tr>
</tbody>
</table>

Table 7: Transition Matrix (50% cum dev)

<table>
<thead>
<tr>
<th>( t \rightarrow t + 1 )</th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>83.83%</td>
<td>16.17%</td>
<td>0%</td>
</tr>
<tr>
<td>Medium</td>
<td>4.17%</td>
<td>86.64%</td>
<td>9.19%</td>
</tr>
<tr>
<td>High</td>
<td>0%</td>
<td>7.92%</td>
<td>92.08%</td>
</tr>
</tbody>
</table>

Table 8: Transition Matrix (All Activities)

Table 8 shows the transition matrix for a developer when she participates in both types of communication and has achieved 50% code development. As expected participation in all the learning activities has a huge impact on moving developers to or keeping developers in the high state.

7.5. Accumulation and Depreciation of Learning

The matrixes in Table 9 are obtained using the Chapman-Kolmogorov Equations. If a developer in low learning state actively participates in all learning activities at an average level continuously for 6 months then she has a probability of 13.54% to be in the high state in the 6th month. For the medium and high
learning states, the probabilities of being in high state after 6 months of continuous participation in learning activities are 33.69% and 68.79% respectively.

<table>
<thead>
<tr>
<th></th>
<th>Active for 6 months</th>
<th>Inactive for 6 months</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t \rightarrow t + 1$</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>Low</td>
<td>39.96%</td>
<td>46.50%</td>
</tr>
<tr>
<td>Medium</td>
<td>11.90%</td>
<td>54.41%</td>
</tr>
<tr>
<td>High</td>
<td>2.91%</td>
<td>28.31%</td>
</tr>
</tbody>
</table>

If a developer in high state does not participate in any of the activities for 6 continuous months then she has a probability of 60.36% of moving to a lower state (specifically 17.90% for low state and 42.46% for medium state). Similarly, continuous inactivity for 6 months on part of a developer in medium state can lead to a probability of 46.39% of being in low state. Note, the probability of being in high state is lower than that of being in low state. This confirms the persisting but dissipating effects of learning as argued by related research (Darr et al 1995).

### 7.6. Posterior Analysis of Individual Behaviors

A developer’s learning 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:

$$P(S_t = s | O_1, O_2, \ldots, O_n) = \pi(i)A(i,1)Q(i,1,2)A(i,2)Q(i,2,3) \cdots Q(i,t-1,t)A(i,t)/L(O_1, O_2, \ldots, O_n).$$

(5)

Here, $Q(i,t-1,t)$ represents the column of the transition matrix $Q(i,t-1,t)$ corresponding to the state $s$. $L(O_1, O_2, \ldots, O_n)$ is the likelihood of the observed outcome sequence up to time $t$.

In any given period, an individual developer can be classified into a specific state according to posterior probability calculation, as described in Equation (5). 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. About 55–60% of the developers at any time are in the low state. Medium state accounts for roughly 20–25%, and the remaining developers are in high state over the time horizon.

---

7. Here the developer participates in average level of activities for the continuous 6 months.

8. This transition matrix is the 6 step transition matrix from transition matrix in Table 5.
under study. This observation is confirmed in Figure 2(b) which shows that the average state is about 1.6–1.7.

![Figure 2: State Distribution of Developers (a) and Average State (b) against Time](image)

At the individual level (Figure 3), we find various types of behaviors. Some developers move reasonable quickly to the highest state, some take quite some time to make the transition and others never move to the highest state. One interesting observation that can be made from Figure 3 is that once a developer transitions to a state she stays there for quite some time before transitioning back.

![Figure 3: State of Individual Developer against Time](image)

### 7.7. Robustness Checks

Table 4 presents a reduced model of our original model which included several other contribution, project and developer specific controls. We controlled for other project specific variables such as programming language, user interface and operating system. CVS commits that deal with changes to an existing file may require more/less effort than the ones that deal with new files. To account for these differences in the type of contribution, we computed a variable that represented a developer’s fraction of CVS commits that
involved changes to existing files for each period. We also employed cumulative communication measures for learning. The effects of these variables were insignificant across all the three states. Of the 251 developers, 162 developers have reported their expertise in the concerned software development area (on a scale of 1 to 5, 5 being the highest) at the time of joining the project. Controlling for this effect does not produce significantly different results. Hence, we present results for only the reduced model.

One concern regarding our results is the appropriateness of period length specification. We employed alternative lengths for a period (2 months and 3 months) and re-performed the analysis. These alternate specifications did not produce qualitatively different results. Similar concern could be raised about the construction of Availability variable. We employed several different constructions of this variable (presence in period $t$ or/and $t - 1$, presence in period $t$ or/and $t - 1$ or/and $t - 2$). Though, the results for other variables were not substantively different, the use of presence in period $t$ as Availability in Period $t$ provides the best likelihood for the parameter associated with it.

7.8. Limitations and Future Research Directions

There are several limitations and we do not want to overstate our findings. First, the latent state concept is an effective method to examine whether participation in code development and peer interactions improve developer productivity and learning or not, but the latency of the states does not allow us to determine the exact impact of learning activities on each of the latent factors. In the absence of such determination it is not possible to separate out the effects on enhancement of DK and PKS from intrinsic motivation. Future research can try to address this limitation through controlled experiment design. Second limitation is that we focus on a very important but a single aspect of learning – learning to contribute code. Future research can investigate other aspects of learning in OSS projects such as learning about better managing the project or process and quality improvements and how much learning from OSS is transferable to commercial development projects. Finally, a simple replication of our research though enriched with more detailed measures on aspects of peer interactions could potentially provide more insights. Each of these limitations represents an exciting area for future research.
8. Conclusions

In this paper, we develop a dynamic model of developer learning in OSS projects. We consider two modes of learning: own experience and peer interactions. Publicly available data from CVS repositories and mailing lists are used for model estimation. This allows us to identify three states which are found to be increasing in both the learning and code contribution probabilities. We also examine the impacts of the two modes of learning on the transition of developers between these states.

Our findings reveal some very important dynamics of OSS environment. Extant research has found that most of the code (approx 80%) is contributed by approx 20% of developers in OSS (Mockus 2002). The major contributors are called the core and the others peripheral developers. A long standing debate among the OSS community deals with the issues: Do initial endowments of a developer have a temporarily persistent effect on their code contributions? (i.e. does the heterogeneity among initial endowments of developers separate them into core and peripheral developers?) Or Are the effects of initial endowments attenuated or accentuated by a developer’s subsequent participation in the project (Cox 1998). Our results reveal that a developer with low levels of initial endowments can learn through subsequent participation in the project and evolve into a core developer over time.

Our results have several implications for attracting and sustaining developer contributions in OSS projects. Our findings reveal that participations in software development activities do not provide just a short term incentive to contribute code, but have an impact on the behavior of a developer while making code contributions. Intense involvement in such activities has the potential to move a developer to a behavioral state which is more conducive to making code contributions. Once a developer is engaged in a certain behavior, such behavior persists. The transition matrices indicate that, for a developer, sustaining the same behavior is considerably easier than changing it. This finding has mixed messages. The positive message implies that a developer with more productive behavior is likely to persist with it without extraordinary effort. The negative message is that a developer engaged in less productive behavior would require loads of effort to switch to a more productive behavior. Based on our findings a manager can devise strategies to help transition the developers to a state which is more conducive to code contributions to the project.
Our findings also reveal that a developer in the highest state sustains such behavior by participating in discussions started by others. An investigation of a select sample of threads participated in by developers in the highest state revealed that the highest state developers were involved in solving others’ problems. Anecdotal evidence also suggests that more advanced developers benefit by helping out others in interesting and intriguing problems. As senior OSS developers Raymond and Moen (2006) state “If you give us an interesting question to chew on we'll be grateful to you; good questions are a stimulus and a gift.” This finding is quite optimistic for the OSS community as the most productive behavior of developer appears to be self sustaining. These results point out that the main managerial focus should be devoted towards influencing the behavior of developers in the low and middle states.

When in the low state developers depend exclusively on their peers to learn. Hence, their code contributions are dependent upon the cooperativeness of peers. Developers in the low state learn the most by participating in threads started by others. However, there is subtle difference between the threads participated in by developers in the low and the high states. We found that the threads participated in by developers in the low state were directed at getting help (compared to providing help by high state developers) for their same or closely related issues. Raymond and Moen (2006) point out that more advanced developer have a “reputation for meeting simple questions with what looks like hostility and arrogance.” However, once a developer moves to a medium or the high state she can draw upon her experience besides that of peers and make significant code contributions. Managerial initiative should be aimed at ensuring that novice or developers in the low and medium state receive the help that would ensure their transition to higher states.

This study makes several contributions. First, this is the first study that investigates learning in OSS environment. Second, this is the first study to model dynamics of developer productivity behavior. We have shown that past experience or peer interactions not only affect the productivity directly but also shape a developer's overall behavior towards contributing code. This modeling framework is a significant addition to the learning curve literature (Argote et al 1990). Third, we add on to the emerging literature in OSS that investigates factors behind project success (Grewal et al 2006, Singh 2007, Singh et al 2007). Amount of code produced has been a measure of project success in all these studies. Our study shows that
effective peer interactions play an important role in the amount of code produced by each developer. Finally, whereas prior research that investigates learning in software development does not account for depreciation in learning, we provide a theoretical basis for such and account for it in our model. Though prior research is silent on the mode of knowledge transfer while learning from peers, we find that the amount of learning from peers in OSS development is a function of the amount of use as well as different aspects of the mode (peer interactions). Taken as a whole, the results support an interesting picture of developer learning in OSS and highlight many avenues for future research to expand our understanding of the unique context of OSS development.

References


Mockus, A., R. Fielding, J. Herbsleb. 2002. Two Case Studies of open Source Software Development: 


Raymond, E.S. 1998. The Cathedral and the Bazaar. *First Monday* 3(3)


Reagans, R., L. Argote, D. Brooks. 2005. Individual Experience and Experience Working Together: 

Roberts, J., I-H. Hann, S. Slaughter. 2006. Understanding the Motivations, Participation, and 


Rossi, C., A. Bonaccorsi. 2005. Intrinsic vs. Extrinsic Incentives in Profit-Oriented Firms Supplying 
Open Source Products and Services. *First Monday* 10(5).

*Neural Computation* 11 305–345.


Appendix

1. Mean Number of CVS Commits for Each State

The mean values for the (rescaled) variables that enter in $W$ are: Software Development = 0.528, Technical Audience = 0.847, Project Rank = 0.029, Manager = 0.324, Involvement Quotient = 0.110, Availability = 0.070, Project Age = 0.070, Project Age Squared = 0.038.

The expected value of number of CVS commits in states 1, 2 and 3 is calculated as follows:

for state 1:

$$\Pr(\eta_1) \exp(\rho_1 W_a + c_1 \eta_1) + \Pr(\eta_2) \exp(\rho_1 W_a + c_2 \eta_2) + \Pr(\eta_3) \exp(\rho_1 W_a + c_3 \eta_3)$$

$$+ \Pr(\eta_4) \exp(\rho_1 W_a + c_4 \eta_4) = 0.0023;$$

state 2:

$$\Pr(\eta_1) \exp(\rho_2 W_a + c_1 \eta_1) + \Pr(\eta_2) \exp(\rho_2 W_a + c_2 \eta_2) + \Pr(\eta_3) \exp(\rho_2 W_a + c_3 \eta_3)$$

$$+ \Pr(\eta_4) \exp(\rho_2 W_a + c_4 \eta_4) = 3.34;$$

and state 3:

$$\Pr(\eta_1) \exp(\rho_3 W_a + c_1 \eta_1) + \Pr(\eta_2) \exp(\rho_3 W_a + c_2 \eta_2) + \Pr(\eta_3) \exp(\rho_3 W_a + c_3 \eta_3)$$

$$+ \Pr(\eta_4) \exp(\rho_3 W_a + c_4 \eta_4) = 42.44.$$ 

2. State Transition Probabilities

The support points and the probability of ($\xi$) are

<table>
<thead>
<tr>
<th>Support</th>
<th>0</th>
<th>0.4306</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probability</td>
<td>0.4697</td>
<td>0.2789</td>
<td>0.2514</td>
</tr>
</tbody>
</table>

The intrinsic propensity to transition from state 1 to 2 is calculated as:

$$\begin{align*}
&\left[1 - \exp\left(\mu(2,1) - c_1 \xi_1\right) \right] \Pr(\xi_1) + \left[1 - \exp\left(\mu(2,1) - c_3 \xi_3\right) \right] \Pr(\xi_3) \\
&+ \left[1 - \exp\left(\mu(2,1) - c_4 \xi_4\right) \right] \Pr(\xi_4) = 0.044;
\end{align*}$$

from state 2 to 3:
\[
\begin{align*}
\Pr(\xi_3) &= \frac{\exp(\mu(3,2)-c\xi_1)}{1 + \exp(\mu(3,2)-c\xi_3)} \Pr(\xi_1) + \frac{\exp(\mu(3,2)-c\xi_1)}{1 + \exp(\mu(3,2)-c\xi_2)} \Pr(\xi_2) \\
&\quad + \frac{\exp(\mu(3,2)-c\xi_3)}{1 + \exp(\mu(3,2)-c\xi_3)} \Pr(\xi_3) = 0.0297; \\
\end{align*}
\]
from state 2 to 1:
\[
\begin{align*}
\Pr(\xi_1) &= \frac{\exp(\mu(1,2)-c\xi_1)}{1 + \exp(\mu(1,2)-c\xi_3)} \Pr(\xi_1) + \frac{\exp(\mu(1,2)-c\xi_1)}{1 + \exp(\mu(1,2)-c\xi_2)} \Pr(\xi_2) \\
&\quad + \frac{\exp(\mu(1,2)-c\xi_3)}{1 + \exp(\mu(1,2)-c\xi_3)} \Pr(\xi_3) = 0.1258; \\
\end{align*}
\]
and from state 3 to 2:
\[
\begin{align*}
\Pr(\xi_2) &= \frac{\exp(\mu(2,3)-c\xi_1)}{1 + \exp(\mu(2,3)-c\xi_3)} \Pr(\xi_1) + \frac{\exp(\mu(2,3)-c\xi_1)}{1 + \exp(\mu(2,3)-c\xi_2)} \Pr(\xi_2) \\
&\quad + \frac{\exp(\mu(2,3)-c\xi_3)}{1 + \exp(\mu(2,3)-c\xi_3)} \Pr(\xi_3) = 0.1606.
\end{align*}
\]