Optimal Group Size for Software Change Tasks: A Social Information Foraging Perspective

Tanmay Bhowmik, *Student Member, IEEE*, Nan Niu, *Senior Member, IEEE*, Wentao Wang, *Student Member, IEEE*, Jing-Ru C. Cheng, Ling Li, and Xiongfei Cao

***Abstract*—Group size is a key factor in collaborative software development and many other cybernetic applications where task assignments are important. While methods exist to estimate its value for proprietary projects, little is known about how group size affects distributed and decentralized cybernetic applications and in particular open source software (OSS) development. This paper presents a novel approach in which we frame developers' collective resolution of OSS change tasks as a social information foraging problem. This new perspective enables us to predict the optimal group size and quantify group size's effect on individual performance. We test the theory with data mined from two projects: 1) Firefox and 2) Mylyn. This paper not only uncovers the mismatch of optimal and actual group sizes, but also reveals the association of optimality with improved productivity. In addition, the social-level productivity gain is observed as project evolves. We show this paper's impact by extending the frontiers of knowledge in two areas: 1) social coding and 2) recommendation systems.**

***Index Terms*—Cybernetic application, group size, productivity, social information foraging theory, task assignment.**

I. Introduction

IN COLLABORATIVE software engineering and distributed cybernetic applications [1]–[5], group size matters. While larger groups are reported to decrease software development productivity [6], smaller ones may lack the problem solving expertise for complex projects [7]. Clearly, the group size affects both performance of individual developers and outcome of the group as a whole.

For traditional proprietary software development, managers must carefully plan and control the project staffing. To support this, many methods are proposed in the software effort estimation literature. For instance, given estimates of development effort in person-hours and software size in function points, an optimal group size can be obtained by regression analysis or Bayesian inference [7]. To balance multiple and often competing objectives, search-based optimization techniques are recently used to assess staffing needs in the presence of schedule fragmentation and communication overhead [8].[1] These approaches assist project managers in determining development team size, matching developer skills, arranging organizational structures, and making other important resource allocation decisions.

By contrast, open source software (OSS) projects rely largely on community participation, and have no formally preassigned effort estimation or control structure. Developers in these projects are grouped organically[2] and dynamically. Research shows that large, successful, and long-lived OSS projects are self-organizing in that developer subgroups spontaneously arise and such groupings manifest strongly in technical collaborations related to software change tasks [10]. Each change task, whether carried out individually or collectively, is aimed at fulfilling some specific goal, e.g., fix a bug, add a functional capability, or enhance a quality attribute. Because these tasks are crucial cogs in the development process machine, how they are performed will have a significant impact on the success of the software project [11].

The information-intensive nature of software change tasks was made evident by Ko *et al.* [12] who showed developers spend much time searching, relating, and collecting relevant information necessary for eventually implementing a solution. Their work was among the first to frame software change as an information foraging problem. Pirolli's information foraging theory [13] uses our animal ancestors' "built-in" food-foraging mechanisms [14] to understand human information seeking and gathering in the vastness of the Web. By modeling software developer as predator and relevant information as prey, researchers were able to better understand developer's behavior in debugging, requirements tracing, and other information-intensive activities [15]–[19], and further suggest tool enhancements in a principled manner [20].

Manuscript received December 29, 2014; revised March 25, 2015; accepted April 2, 2015. Date of current version July 15, 2016. This work was supported by the U.S. National Science Foundation under Grant CCF-1350487. This paper was recommended by Associate Editor L. D. Xu.

T. Bhowmik is with the Department of Computer Science and Engineering, Mississippi State University, Mississippi State, MS 39759 USA.

N. Niu and W. Wang are with the Department of Electrical Engineering and Computing Systems, University of Cincinnati, Cincinnati, OH, 45221 USA.

J.-R. C. Cheng is with the Information Technology Laboratory, U.S. Army Engineer Research and Development Center, Vicksburg, MS 39180 USA.

L. Li is with the Department of Information Technology and Decision Sciences, Old Dominion University, Norfolk, VA 23529 USA.

X. Cao is with the School of Management, University of Science and Technology of China, Hefei 230026, China (e-mail: caoxf312@126.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2015.2420316

[1] Brooks, in his seminal work *The Mythical Man-Month* [9], noted there is no simple linear relationship between the number of developers and the engineering time required for a project. This is eloquently stated in Brooks's law: "Adding manpower to a late software project makes it later." Di Penta *et al.* [8] showed that the impact of Brooks's law on project staffing is subtle and could be contained.

[2] By organically, we mean not externally forced but internally developed.

While the current studies confirm foraging theory's applicability and demonstrate its usefulness in software engineering, the focus has been predominantly on tasks performed by a *solo* developer. Many OSS change tasks, however, are accomplished as the result of collective action, in which a group of developers engage in social exchanges of information to make joint contributions. Pirolli [21] proposed the basics of *social* information foraging theory where he presented mathematical models to predict, among other phenomena, the effect of group size on individual member's rate of gain.

In this paper, we extend developers' solo information foraging toward their foraging in groups. We adapt Pirolli's models [21] in the context of OSS change tasks. Each task's discussion and resolution, which we mine from the project's issue tracking system, give rise to a patch of information that yields some amount of utility for one or more foragers (developers). We distinguish the tasks done solitarily from those achieved by collectives, and then identify optimal group size based on the foraging-theoretic predictions [21]. The theoretical characterizations allow us to formulate specific hypotheses regarding the individual rewards of cooperation, as well as the self-organizing aspect of evolving OSS projects. We test our hypotheses by using the data collected from two successful OSS projects: 1) Firefox and 2) Mylyn.

The contributions of this paper lie in the novel perspective that links software developers' rational behaviors [15]–[19] together with their *social* information foraging. This paper is among the biologically inspired approaches to tackling cybernetic challenges [22]–[24]. Our vision in this paper is to transform foraging theory's ecological validity and predictive accuracy [13], [14] to establish a robust grounding for studying a wide range of software engineering events and activities. We articulate this vision by illuminating our work's potential impact on two research fronts: social coding [25] and recommendation systems [26]. In what follows, we survey related work in Section II. We then present our research method and hypotheses in Section III, and describe the empirical analysis results in Section IV. The implications of this paper are discussed in Section V, and finally, Section VI concludes this paper.

II. Background and Related Work

This paper builds on the foraging-theoretic relation between the size of a group and the group member's rate of gain [21]. As software change task is our primary concern, we correspond rate of gain to productivity which helps to assess developers' gain of useful information to their tasks per unit time. This section begins with the preliminaries of social information foraging, and then reviews the software engineering literature related to group size and productivity.

A. Information Foraging Theory

Pirolli developed information foraging theory as an ecological-evolutionary approach to understanding users' information seeking on the Web [13]. The general idea is that we can scientifically study human and technological adaptations to the flux of information in the social environment in the same way as biological adaptations to the flux of energy in the physical environment.

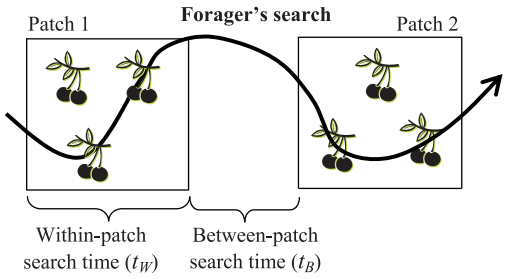


Fig. 1. Illustration of foraging in a patchy environment.

Information foraging, then, is derived from optimal foraging theory in biology, which analyzes the adaptive value of food-foraging strategies [14]. Optimality here refers to the strategy that maximizes the gain per unit time of foraging. Fig. 1 illustrates the elementary constructs by presenting a hypothetical bird's foraging in an environment that consists of berry patches. The forager must expend some amount of between-patch time (t_B) arriving at a patch, and t_W denotes the within-patch foraging time. Thus the rate of gain is

$$R = \frac{G}{t_B + t_W} \tag{1}$$

where G represents the expected net gain. By mapping the constructs to Web navigation (e.g., each webpage is considered as an information patch) and applying the core mathematics like (1), Pirolli modeled an optimal Web user's behavior [13]. This provides remarkable insights into issues like link selection and decision to leave a webpage. As a result, information foraging theory has become very useful as a practical tool for website design and evaluation [27].

Inspired by human's adaptive interaction with information on the Web, researchers began to apply foraging theory in software engineering. Notably, the pioneering work by Lawrance *et al.* [15], [16] showed encouraging results matching foraging theory's predictions with real developers' behaviors in debugging. Other studies (including our own) widened the theory's scope of applicability from requirements and architecture to refactoring and reuse [17]–[20]. Common to all the studies is the key role played by cues. Cues, such as call dependencies and lexical similarities in the code base, are signposts that exist only in the environment [16]. Meanwhile the cues can be annotated, decorated, or otherwise brought to attention for the predator (i.e., software developer) to improve the foraging efficiency [20].

So far, foraging theory has mainly focused on information seeking by the solitary developer [15]–[20]. However, today's software (especially OSS) is rarely developed by soloists but is the result of collective efforts. Drawing on the quantitative theories of cooperative problem solving [28] and group foraging [29], Pirolli extended information foraging to the social level [21]. The key assumption is that cues are exchanged in social information foraging regarding the likely location of

useful information. Apart from the cues perceived in the environment, foragers can benefit from the hints shared by group members (e.g., where to find what information) so as to better achieve their individual goals.

Although there are positive effects of social foraging, foraging groups do not become arbitrarily large, suggesting that there exist interference costs (e.g., communication overhead in large software projects [8], [9]) that at some point outweigh the advantages of further increments in the group size. In OSS development, the study by Hong *et al.* [30] mined Mozilla's change history from 2000 to 2009, and showed the number of active developers was stable over time. Similar phenomena can be empirically observed in Wikipedia whose creation and maintenance, like OSS projects, depend largely on social participation. It was shown that the number of editors actively contributing to Wikipedia had plateaued [31]. Note that we study group size at the task level in this paper, not at project [31] or subcommunity [30] levels.

The cost-performance tradeoff of group foraging can be formalized as follows [21]. Let us assume that the individual forager's time to process an information patch in a group of n foragers is: $\tau(n) = an^c$, where $0 < c < 1$ is a rate parameter and a is the time to forage for a patch when $n = 1$. The expected gain for each group member is updated to be G/n. If λ denotes the individual search rate, then the group rate is $n \cdot \lambda(n)$. The interference time can be modeled as $t_I = 1/[n \cdot \lambda(n)]$ [29]. Furthermore, let $\lambda(H)$ denote the rate of finding valuable information patches with H distinct hints. Then the expected time for n foragers to encounter a valuable patch is $t_B = \lambda(H)/[n \cdot \lambda(n)]$. Finally, when n predators forage simultaneously, the patch is exhausted in $t_W = \tau(n)/[n \cdot \lambda(n)]$ time units. We may now cast group foraging as a variation of the conventional model presented in (1). Hence, the rate of gain for the individual member of the group is

$$R(n, H) = \frac{G/n}{t_I + t_B + t_W} = \frac{G/n}{\frac{1}{n\cdot\lambda(n)} + \frac{\lambda(H)}{n\cdot\lambda(n)} + \frac{\tau(n)}{n\cdot\lambda(n)}} = \frac{\lambda(n) \cdot G}{1 + \lambda(H) + \tau(n)}. \quad (2)$$

The conceptual illustration of (2) is presented in Fig. 2. Using group size as an approximation of hint diversity ($n = H$), one can see a basic lognormal distribution with the peak value $n^\star$ that theoretically defines the optimal group size. Compared to the rate of return for solitary foraging $R(1, 1)$, the group of size $n^\star$ best manifests the power of cooperation as the potential for every member to find useful information, thereby to reach their individual goals, is maximized.

In sum, foraging theory stems from the assumption that all organisms (including humans) are ecologically rational and adapt to the environments in which they operate [13], [14]. Applying the theory in software engineering has been particularly fruitful in understanding how solo developer performs information-intensive tasks. Extending the theory to the social level not only coherently connects actions and interactions among developers, but also quantitatively characterizes the foraging-theoretic limit on individual performance imposed by group size.

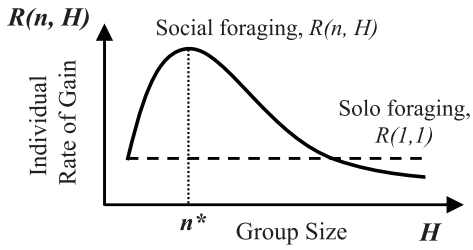


Fig. 2. Lognormal distribution of (2) adopted from [21]. Assuming $n = H$, the peak value of $R(n, H)$ gives rise to the optimal group size $n^\star$. The dashed horizontal baseline, $R(1, 1)$, shows the solo forager's rate of gain.

B. Group Size

Group size plays an important role in software development. Some methods explicitly prescribe its value range. For example, the agile practices best suit colocated teams of about 50 people or fewer [32], and the team software process is designed for use with teams of 2 to 20 members [33].

Problems arise if the development team is too small or too large. Small teams may not be equipped with the diverse expertise required to solve complex tasks [7]. Moreover, a small team can lead to a heavy bias in coding and testing, which incurs a high maintenance cost. Microsoft, for instance, used a strategy of employing small teams of star developers and found that the strategy, when confronted with the maintenance of large mass-marketed applications, did not work well [34]. On the other hand, developers in larger teams can become less motivated and productive, encounter more conflicts and coordination difficulties, and experience increased risks of social loafing and free riding [6], [9], [35], [36].

Optimal group size, thus, has attracted much software engineering research attention. As early as 1978, Putnam's definition considered group size to be optimal if it allows the developers to achieve the maximum productivity with shortest schedule and lowest cost without affecting the final outcome [37]. Various approaches have been presented to forecast the number of developers a proprietary software project should have. These approaches fall into four categories.

1) *Empirical:* Putnam sampled 491 projects and concluded that productivity is higher for smaller teams with an optimal group size of 3–5 staffs [37]. The rule of thumb—teams of nine or more are significantly less productive than smaller teams—defines another empirical threshold which is supported by studies like [35] and [38].
2) *Analytical:* Methods in this category assume certain functional forms in order to establish a relationship between group size and other project variables. Examples include an opening hyperbola model [39], approximating the Cobb–Douglas production function [40]: development effort $= A \cdot (\text{software size})^b \cdot (\text{group size})^c$, where A and b are parameters that take positive values, and c is a nonnegative constant exponent.
3) *Probabilistic:* Causal models are often built by using supervised machine learning techniques like Bayesian networks [41]. Although such models are susceptible to overfitting, they allow probability bounds to

be established and integrated into posterior estimates. For instance, the forecasted effort for the experimental dataset in [41] was 8.28 person-months with 94% chance that the actual effort would be less than 20 person-months.

4) *Search-Based:* These approaches employ metaheuristic techniques to find (near) optimal solutions related to group size. Abdel-Hamid [42] developed a system dynamics model for staffing estimation, and applied the model in NASA's DE-A project to analyze the decision of allocating up to eight people in one team. Antoniol *et al.* [43] exploited queuing simulation along with multiobjective optimization, and found that 46 was the optimal staffing level for a large maintenance project.

Several points are worth noting. First, hybrid estimation method exists [7]. Second, while the majority of studies confirmed the effect of group size on software development effort, Smith *et al.* [44] showed the effect was not always statistically significant. Finally, optimality does not mean sticking to the same group size over the entire project lifespan. In fact, search-based approaches [8], [42], [43] provide dynamic restaffing capabilities so that managers can better allocate project resources at different stages of the software life cycle.

Compared to formally managing staffing levels in closed-source software development, community contributions are the life's blood of a successful OSS project. The subgroup structure emerges gradually and organically based on how developers communicate and collaborate with each other [10], [30], [45], [46]. The case study on Mozilla showed that the size varied greatly from one group to another [30]. As Mozilla evolved in the 2000s, the size of the median group fluctuated between 14 and 141 developers. What is not known is to what extent and in which way the group size affects individual performance. We believe social information foraging provides a direct answer. This answer, in turn, can shed light on the self-organizing aspect of successful OSS projects, as well as developers' rational behaviors in the autonomous social groups.

In sum, managing the group size in proprietary software projects involves an intriguing paradox: while larger teams clearly invest more human capital, smaller ones seem to produce better teamwork. Although traditional methods in software effort estimation are less suitable for studying OSS projects, the relation between group size and developer productivity spelled out in Putnam's early definition [37] remains essential to our inquiry.

C. Developer Productivity

Definitions of productivity share such common elements as efficiency, input, and output. As one example, the IEEE 1045 Standard[3] defines productivity in terms of the rate of output per unit of input. For software, source code is among the most tangible outputs, and the input unit is often based on time. While conforming to the IEEE Standard in principle, the literature contains many productivity measures [47], e.g., number of lines of code (LOC) per person-month, number of LOC per hour, etc.

Despite all the measures, little is known about developers' own perceptions of productivity. Meyer *et al.* [48] recently filled the gap by surveying 379 software professionals and observing 11 developers at work. The results indicate that developers often reflect productivity in days. They perceive their days as productive when many or big tasks are completed. It is also emphasized that, regardless of the measure, productivity should be used to enable within-project evolutionary analysis and retrospective improvement, rather than to make direct comparisons across individuals or across organizations [48].

Similar line of research has considered developers' perceptions in OSS development. Dabbish *et al.* [25], [49] investigated productivity implications by examining how GitHub users make social inferences based on the visible cues and signals in the environment. Among the cues, commits—software changes submitted to the project repository—play a critical role [49].

1) Amount of commits implies commitment, liveness, and community attention.
2) Type of commits signals interest in different aspects of the project.
3) Relationship between commits and comments, issues, or other commits conveys intention behind developer actions.
4) History of commits can be used to infer project structure, roles, and developer expertise.

In sum, most developers tend to assess their productivity through the tasks completed [48]. Completing OSS software change tasks typically involves commits, which provide important cues for making social inferences about developer's behavior [25], [49]. Understanding how working with others affects one's own productivity is precisely the focus of this paper.

III. Research Methodology

Our overall research objective is to examine to what extent the optimal group size of social information foraging [see (2) and Fig. 2] holds in OSS change tasks. Compared to such functional forms about group size as [39] and [40], (2) is different because it does not involve software size or other effort estimation variables that are less suited to OSS projects. Instead, the relationship is built on human's collective problem solving [28], and connects explicitly and directly the group size with individual member's performance. As our analysis is both analytical and evolutionary, this section presents how we map theoretical constructs to OSS project data followed by our formulation of specific hypotheses.

A. Project Selection and Data Extraction

Despite the lack of formal mechanisms to control staffing levels, there are OSS projects with sizable developer pools that produce software of high quality and rich functional capabilities that rivals their commercial counterparts. This paper studies two such systems: 1) Firefox and 2) Mylyn. Both are

[3]http://standards.ieee.org/findstds/standard/1045-1992.html

TABLE I
INFORMATION ON THE DATA GATHERED FOR THE SUBJECT PROJECTS

	Firefox	Mylyn
Domain	Web browsing	Task management
Source	mozilla.org/firefox	eclipse.org/mylyn
Programming Language	C/C++, JavaScript	Java
# of Source Code Files	1968 (C/C++)	2321
Analysis Begin Date	2004-07-06	2006-12-05
Analysis End Date	2011-06-20	2011-02-28
# of Unique Developer IDs	2569	149
# of Completed Tasks	2878	1898
# of Commits	18538	4908

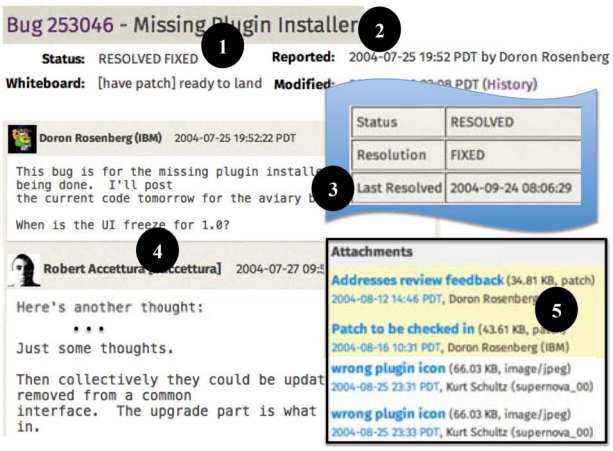


Fig. 3. Sample completed task of Firefox—some contents are omitted, truncated, and rearranged. ❶: task's status. ❷: task's opening (reported) time. ❸: task's closing (last resolved) time. ❹: comments. ❺: commits.

stable and successful in their respective domains. Each has undergone a number of major release cycles and is still under active development. Table I provides some general project information, as well as the data collected for our analysis.

We have selected projects that vary in their governance structure [10] and task[4] triage process [50]. Firefox is a foundation project and follows a volunteer-based triage process. Mylyn, though started in a monarchy way as part of Mik Kersten's Ph.D. thesis, gradually evolves to a community centering around the open source implementation of the task-focused interface. For Mylyn, determining the relevance and priority of each submitted issue is developer-based. With the variety in these different dimensions, our intention is to ameliorate some of the threats to external validity.

For each project, we extract the successfully completed tasks whose opening and closing times fall into our analysis period. We then classify the tasks based on the discussion and resolution information recorded in the project's repository. Fig. 3 shows an example. We define a task is solo if only one developer is involved in the task's whole life cycle, i.e., from task reporting, through commenting and committing, to its final resolution. Otherwise, the task is social due to developers' interaction. Fig. 3 illustrates a social task's collective problem solving where multiple developers exchange

[4]We use "task," "issue," and "bug" interchangeably in this paper to refer to the software change tracked in systems like concurrent versioning system and Bugzilla.

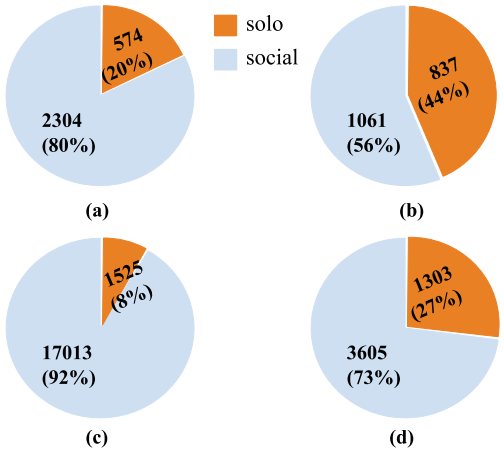


Fig. 4. Solo-social distinctions. (a) Firefox and (b) Mylyn tasks. (c) Firefox and (d) Mylyn commits.

comments and submit commits. Fig. 4 displays solo-social proportions of the collected tasks and commits. While a majority of the activities are social, the dominance is more prominent in Firefox than Mylyn.

To examine the effect of group size, one must determine a unit of analysis (i.e., what counts as a group). On one end of the spectrum, every task can single out a group. On the other end, the entire project group can be analyzed as a whole. We decide to create a series of developer networks and treat each network as a social group in our analysis. The purpose of such networks is to represent the nexus of socio-technical relationships between developers in a software project [45]. Here, "socio-technical" refers to the connection between two people in the context of work-related collaboration [51]. We carefully reviewed some recent and representative approaches to developer social network analysis. Table II lists these approaches' characteristics and also positions our construction in the relevant literature.

We consider all developers in this paper rather than just the core members with commit right [10], [30], [45]. If a person participates in task discussions like Robert in Fig. 3-❹, we think it is a contribution to the collective problem solving. In addition, Robert may be able to submit commits in the future even if his write access is not granted now. Two developers[5] are connected in our network if they work on the same task by exchanging cues—comments or commits—in their social information foraging. The commits can be code or other files (design, testing, etc.) like the icon image in Fig. 3-❺. A time window of three months is used to aggregate the work-related collaborations into a single data point for our group-size analysis. In choosing the three-month window, we want to balance the ephemeral nature of collaboration [45] and the level of stability observed in OSS project's social structure [10] and evolution [30].

We therefore obtain 28 developer networks (data points) for Firefox and 17 for Mylyn with the last network of both

[5]While aliasing presents a challenge, especially for building e-mail social networks [10], we believe its impact is minor in this paper because one developer is unlikely to use different accounts during the resolution of the same task. We thus leave the investigation of aliasing's influence to future work.

TABLE II
SOFTWARE DEVELOPERS' SOCIAL NETWORKS

Developer Network	Nodes	Edges (n1→n2)	Edge-Window	Studied Project(s)
E-mail network [10]	Developers with commit right	n2 replies n1's e-mail	3 months	Apache, Ant, Python, Perl, PostgreSQL
Communication network [52]	All developers	n2 reads the information provided by n1	between builds	IBM's Jazz
Tesseract [53]	All developers	n1 and n2 edit the same source code file	user-selected time period	GNOME
Codebook [54]	People and artifacts (9 node types)	befriend (18 edge types)	6 months	Data based on a medium-sized Microsoft team
Collaboration network [55]	All developers	n1 and n2 are co-listed as con -tributors of the same project	N/A	SourceForge (Sept 2009 data dump)
DPP tripartite graph [56]	Developers who contribute to >7 projects	n1 and n2 work on similar projects	N/A	SourceForge (May 2008 – May 2010 data dumps)
Socio-technical network [45]	Core developers (>50 commits in 2 years)	n1 and n2 make a commit to the same source code file	1 month	Linux kernel, PHP, Wireshark
Subcommunity evolution [30]	Core developers (edge weight > 2)	n1 and n2 work on the same task	6 months	Mozilla
Our work	All developers	n1 and n2 work on the same task	3 months	Firefox, Mylyn

projects building on collaborations less than three months. If a task spans our three-month window boundary, we classify it according to its closing time and not its opening time. This decision is made for accurately counting the number of successfully completed tasks in each time window. We now can instantiate the parameters in (2) in order to calculate the optimal group size $n^\star$ predicted by social information foraging theory.

1) We map each task as an information patch in which solo or social information foraging occurs.
2) Following Pirolli [21], we assume $n = H$. We further assume the amount of within-patch information gain, G, equals to the number of effective hints, H. Therefore, $n = H = G$.
3) The group rate of finding useful information is $\lambda(n) = \lambda(H) = 1/t_{\text{Patch}}$, where t_{Patch} is the within-patch foraging time [21]. In our case, t_{Patch} = [task's closing time − task's opening time] (see ❷, ❸ in Fig. 3).
4) To compute the patch processing time $\tau(n) = an^c$, we assign the value of a (solo foraging time) to be the average t_{Patch} of all the solo tasks inside each time window. The rate parameter c is then calibrated to obtain the best possible lognormal curve for $R(n, H)$ [21]. For our collected data, $c = 0.3$ in both Firefox and Mylyn.

It should be emphasized that the purpose of $R(n, H)$ is not to estimate developer's actual productivity, but to theoretically determine $n^\star$ (optimal group size). Our primary goal of leveraging the $R(n, H)$ model (see Fig. 2) is twofold: 1) to provide a mathematical rationale for and quantitative insight to the key tradeoff in OSS development (i.e., developer groups do not become arbitrarily large [30]) and 2) to allow for critical comparisons between the theoretically predicted $n^\star$ and the actual group size n observed empirically.

B. Hypotheses

Our first testable hypothesis concerns the degree to which $n^\star$ and n match with each other.

H_1—There is no difference between $n^\star$ (the optimal group size) and n (the actual group size).

We next test the foraging-theoretic relation between group size and individual group member's rate of information gain. As mentioned earlier, we assess the rate of gain by developer productivity. Based on the related literature (see Section II-C), we measure productivity in OSS development by the number of commits per day divided by the number of tasks performed on that day. For example, if on a particular day, Task_1 is done collectively by Ana (three commits) and Bob (five commits), Task_2 by Ana (five commits) and Chris (one commit), Task_3 done individually by Ana (three commits), Task_4 by Chris (two commits), and Task_5 also by Chris (four commits), then the productivity of Ana_{solo}, $\text{Ana}_{\text{social}}$, $\text{Bob}_{\text{social}}$, $\text{Chris}_{\text{solo}}$, and $\text{Chris}_{\text{social}}$ is $3/1 = 3$, $(3+5)/2 = 4$, $5/1 = 5$, $(2+4)/2 = 3$, and $1/1 = 1$, respectively. This measure thus approximates a developer's gain of useful information in carrying out daily OSS change tasks. According to the social information foraging model [21], we have the following.

H_2—The closer the actual group size is to optimal, the more productive the group members are.

Self-organizing has been recognized as a key for understanding how OSS development groups coordinate themselves and make autonomous decisions. Bird *et al.* [10] noted that latent subcommunities emerge as the OSS project evolves. Interestingly, Hoda *et al.* [57] focused on agile software teams by showing six self-organizing roles which are implicit and spontaneous. We posit that foraging theory offers a new perspective on how social groups self-organize.

H_3—As the OSS project evolves, the group size becomes more optimal.

Our last hypothesis investigates developer's solo-social behavior changes. Such changes are important because they directly impact productivity [58], [59]. For example, a developer may achieve individual productivity gain by reducing social interactions [48]. However, the change should not compromise social-level performance if the group is self-organizing. We therefore formulate our final hypothesis as follows.

H_4—Developer's solo-social changes during the OSS project's evolution lead to productivity gain at the group level.

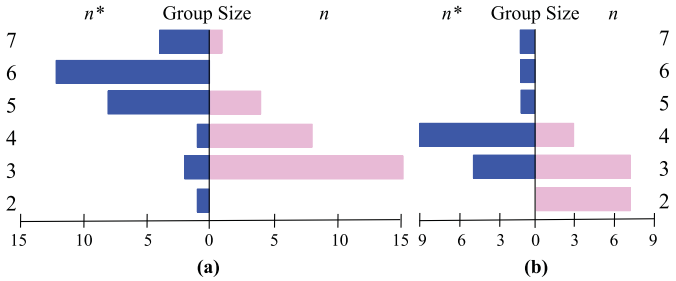


Fig. 5. Group size distribution: each bar represents the frequency of occurrence of $n^\star$ or n in [2, 7]. (a) Firefox. (b) Mylyn.

IV. Empirical Analysis Results

We use the data collected from Firefox and Mylyn to examine the influence of group foraging size both analytically ($\mathbf{H_1}$ and $\mathbf{H_2}$) and evolutionarily ($\mathbf{H_3}$ and $\mathbf{H_4}$). Our statistical inferences are not causal but correlational, as our current interest is in exploring the novel relationships suggested by the theory of social information foraging rather than determining the direction of causality in the relationships.

To evaluate $\mathbf{H_1}$, we perform the Mann–Whitney test [60], a nonparametric test previously used to assess whether information foraging theory's predictions and developers' actual behaviors match with one another [15], [19]. Fig. 5 presents descriptive statistics about optimal ($n^\star$) and actual (n) group sizes in pyramid plots. The determination of $n^\star$ is illustrated in Fig. 6. Although the operationalizations in our current study have certain limitations that will be discussed later in this section, a head-to-head comparison of $n^\star$ and n tests theory's predictions in a direct manner. As can be seen from Fig. 5, even though $n^\star$ and n in both projects fall into the range of [2, 7], their value discrepancies are very noticeable. The Mann–Whitney tests further confirm that the differences between $n^\star$ and n are statistically significant (Firefox: $U = 116.0$, $p < 0.01$; Mylyn: $U = 29.5$, $p < 0.01$). Therefore, $\mathbf{H_1}$ is rejected.

We test $\mathbf{H_2}$ by calculating the association between two variables: $\Delta n = |n^\star - n|$ and ΔP. We define ΔP as the absolute value of the difference of developers' average productivity in solving social tasks and that in solving solo tasks. Following our earlier example in Section III-B, $\Delta P = |(\text{Ana}_{\text{social}} + \text{Bob}_{\text{social}} + \text{Chris}_{\text{social}})/3 - (\text{Ana}_{\text{solo}} + \text{Chris}_{\text{solo}})/2| = |(4 + 5 + 1)/3 - (3 + 3)/2| = 0.33$. For Firefox and Mylyn, 28 and 17 pairs of (Δn, ΔP) are collected, respectively. We then use Spearman's rank correlation coefficient [60], another nonparametric measure, to assess the statistical dependence between Δn and ΔP. The tests of both projects result in negative values of Spearman's ρ at significant levels—Firefox: $\rho = -0.46$, $p < 0.05$; Mylyn: $\rho = -0.63$, $p < 0.01$. This indicates that the increase of ΔP is strongly associated with the decrease of Δn. Thus, $\mathbf{H_2}$ is supported.

The test of $\mathbf{H_3}$ involves the temporal trend analysis of Δn. The analysis is performed along the three-month time series. For illustrative purposes, Fig. 6 plots the lognormal curves of two arbitrarily chosen time windows for each project. One can then theoretically determine $n^\star$ for each time window as shown in Fig. 6. If we assign ID numbers (#1, #2, #3, ...) to the time

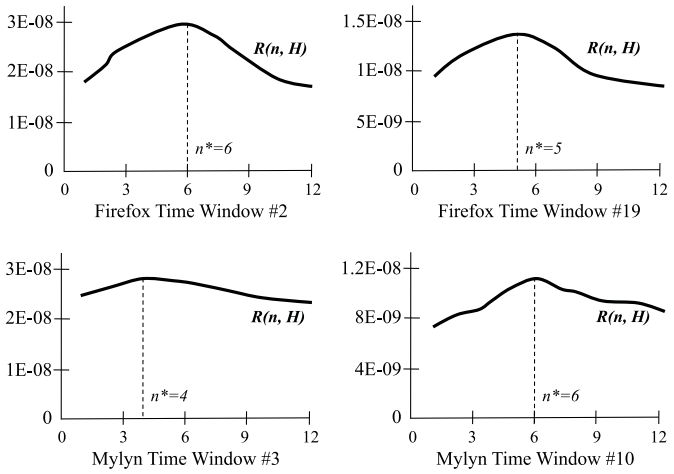


Fig. 6. Determining foraging-theoretic optimal group size $n^\star$. The displayed time windows (units of analysis) are chosen arbitrarily. We measure the within-patch foraging time t_{Patch} in minutes. This measurement unit affects only the absolute scale of $R(n, H)$ as the shape of the lognormal curve and the value of $n^\star$ remain unchanged.

windows, then a strong negative correlation between the time window IDs and Δn provides support for $\mathbf{H_3}$. The negative correlation would indicate that Δn tends to decrease when ID increases (i.e., the project evolves). However, Spearman's tests show positive correlations at statistically insignificant levels (Firefox: $\rho = 0.02$, $p = 0.92$; Mylyn: $\rho = 0.29$, $p = 0.26$). To further visualize the temporal trends, Fig. 7 depicts the value of $(n^\star - n)$ instead of $|n^\star - n|$. For Firefox, the actual group size matches perfectly with the optimal value only once at time window #19. In other times, n is strictly greater or less than $n^\star$. Surprisingly, the actual group size in Mylyn is always less than or equal to the corresponding optimal group size. We speculate that one possible reason might be that different projects or project phases employ different organizational social structures [61] and hence different group formations and sizes; testing this speculation requires future research. Nevertheless, it is clear from Fig. 7 that n and $n^\star$ do not become closer as the project evolves. Due to this observation and Spearman's test results, $\mathbf{H_3}$ is rejected.

Compared to the quantitative analyses of the above hypotheses, $\mathbf{H_4}$ is assessed qualitatively mainly because we could not find systematic and reliable methods in the literature for detecting developer's solo-social behavior changes. We thus use a purposeful sampling strategy [62] by directing our attention to such high-profile developers in the community that their solo-social task-solving behaviors likely have an important influence on the social groups. For Firefox, we focus on its former lead developer, Ben Goodger, who was hired by Google in January 2005 for the Chrome project. For Mylyn, a natural choice to us is its creator Mik Kersten. We then inspect the software change activities of these two developers and select the two consecutive time windows with the most noticeable solo-social behavior changes to test $\mathbf{H_4}$. Fig. 8 summarizes the results.

1) Among all the Firefox project history that we analyzed, Goodger performed the most number of solo and social

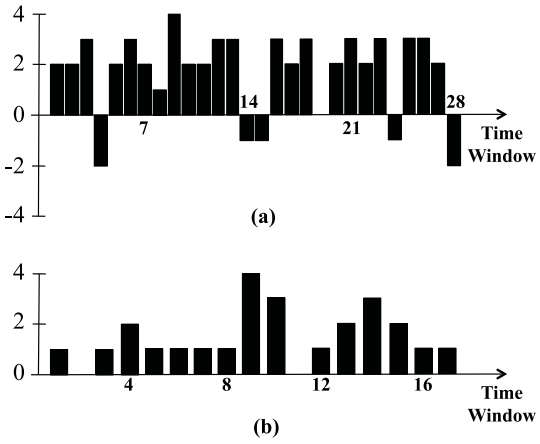


Fig. 7. Plotting $(n^{\star} - n)$ during project's evolution. (a) Firefox. (b) Mylyn.

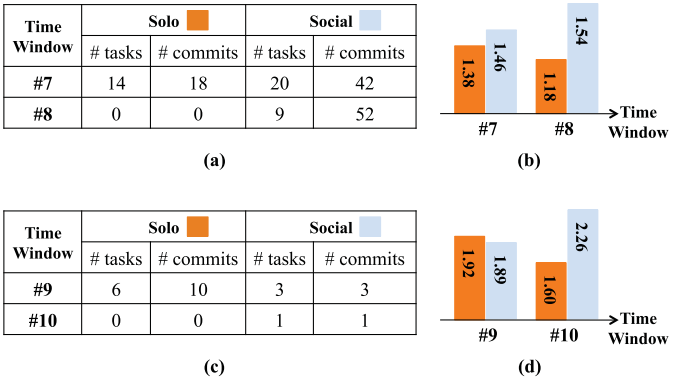


Fig. 8. Solo- and social-level changes as the OSS project evolves. (a) Goodger's and (c) Kersten's individual behavioral changes. (b) Firefox and (d) Mylyn productivity changes.

tasks in time window #7: 14 and 20, respectively. For the next three months, his engagement in software change tasks reduced and the nine tasks that he contributed were all collaborations with other developers. Fig. 8(b) shows that during this evolution, the group-aggregate productivity for solving tasks solitarily decreased whereas the collective task solving productivity improved.

2) The trend of Kersten's changes from Mylyn's time window #9 to #10 is similar to Goodger's: during the project evolution, no further solo task was performed and the number of social tasks was also dropped. Such individual changes, when understood from the standpoint of foraging in groups [see Fig. 8(d)], correlated with social-task productivity gain and solo-task productivity loss.

Our qualitative analyses provide initial evidence that $\mathbf{H_4}$ holds. In both cases, productivity gain of social groups is observed. While Goodger and Kersten might stop working on tasks individually, they still keep social interactions with others in the community. An interesting observation is that the difference between solo and social productivity becomes greater as the project evolves. We consider this to be another facet that indicates how developer groups self-organize, namely by trading soloist's performance off social capital. While both instances follow a shift from solo to social, we must be cautious about how much $\mathbf{H_4}$ findings can be generalized. In other software projects, or even in different time periods of Firefox and Mylyn that our current analysis does not cover, a (lead) developer's social-to-solo shift may as well lead to average productivity gain, possibly due to the high-level expertise involved in resolving software change tasks.

The results of our empirical study can be summarized as follows. The fact that $\mathbf{H_1}$ and $\mathbf{H_3}$ are refuted implies that it is not the group size *per se* that is important. Rather, the support for $\mathbf{H_2}$ and $\mathbf{H_4}$ indicates that we shall go beyond the optimal group size by connecting it to developer productivity, especially in the context of software evolution [63].

A. Threats to Validity

A major limitation is our choice of using a three-month time window to define developers' social groups. While this operationalization refines the six-month window recommended in [30] for understanding OSS project evolution, sensitivity analyzes such as those conducted in [56] can help to reason about the design decision more thoroughly. A related threat is our reliance on discussion and commits information to detect social links between developers. Despite the popular use of the issue tracking data in related studies [30], [45], [52], we miss other potential developer interactions such as private e-mails and chats over internet relay chat channels. This affects our identification of solo tasks more than social ones.

Another limitation relates specifically to our reliance on qualitative analysis for testing $\mathbf{H_4}$. We note that quantitative metrics and approaches could be used. For example, an abnormal return may be measured by the following steps. First, one defines significant events in the software project that are considered important to $\mathbf{H_4}$, e.g., times when top developers decrease their solo activities, or increase their social activities, or both, by a certain amount. Next, compute the variable of interest (in our case $n^{\star} - n$) in a period before the event (from which one makes an expectation) and after the event (where one computes the actual value). Finally, the abnormal return is calculated by the difference between actual and expected, which can be a quantitative indicator used to assess $\mathbf{H_4}$. Note that more operational insights and quantitative measures of $\mathbf{H_4}$ could possibly be derived from Putnam's study on social capital [64].

The way we measure developer productivity poses a threat to construct validity. In fact, no single productivity measure is perceived valid unanimously by developers themselves [48]. It is unlikely such measure will ever exist. Therefore, the number of daily commits normalized by tasks and developers that we use in this paper should be treated only as an approximate. However, we hope the approximate, which is based on the literature review (see Section II-C), is useful not because of the absolute values calculated but the way it is being used in this paper—to support correlational and evolutionary analyzes.

While social information foraging theory asserts the relationship between group size and group member's rate of gain [21], it is important to note that no causal link has been established in our empirical study. Future work is required to determine if the optimal group size drives productivity gain

or vice versa, or if they are both results of some unobserved phenomenon.

The biggest threat is to external validity. As with most OSS studies (see Table II), only a small number of projects could be selected. We chose Firefox and Mylyn because they are mature, stable, long lived, and considered successful. As mentioned earlier, we also tried to incorporate the variability of these two projects' governance structure and task triage process. However, many other project attributes exist and we have no evidence with respect to how much our results will be generalizable to a wider range of projects. We want to point out that Firefox and Mylyn have been studied in prior research (see [30], [50]), allowing the research community to integrate our results with the findings of others.

V. Discussion

Our foraging-theoretic inquiries into optimal group size enable new ways to support software practitioners and organizations. To demonstrate our work's potential impact, we discuss in this section how our results can be applied. Specifically, we show the improved understanding and enhanced support derived from this paper in order to extend the frontiers of knowledge in two software engineering areas.

A. Social Coding

For many years, software development environments have been designed for developers to focus on their own work without much interference. Principles like information hiding [65] and mechanisms like pessimistic version control [66] attempt to free the developer from complexities and inconsistencies resulting from colleagues' actions. Although helpful, the isolation is not ideal for collaborative software development. Several tools, such as Jazz [67], Palantír [68], and Crystal [69], have arisen to answer the collaborative needs by raising change visibility and enhancing conflict management as software evolves.

A more radical approach is now sweeping the OSS world and gradually working its way into corporate environments [70]. The approach, exemplified by GitHub whose tagline is "social coding," aims to dramatically improve the level of collaboration and participation among people who build software [71]. In a nutshell, social coding fuses social networking functionalities with flexible version control systems such as Git, Mercurial, or Bazaar [25]. On one hand, social networking helps create a transparent work environment that allows developers to easily review feeds, watch projects, and follow others. On the other hand, flexible version control systems do away with the idea of a single master branch, allowing developers to discover interesting changes, experiment with them in separate forks, pull others' changes into their own branches, and offer changes back to the repository owner [25].

Because of the flexibility of participation and the power of collaboration, social coding has led to improvements in terms of quantity and quality of work that different communities and companies have done [71]. The level of participation can be understood from the social information foraging perspective as shown in Fig. 9. In the figure, the point where cooperation curve crosses the solo information foraging threshold gives rise to the equilibrium group size [21]. Even though members of the group may see their individual rates of return diminish from the optimum at $n^\star$ as new members join the group, remaining in the group is still better than solitary foraging. Consequently, the effect of reduced collaborative costs (e.g., easier sharing and notification in GitHub [25]) can be linked to the increased equilibrium group size. The quantitative prediction is shown in Fig. 9, in turn, can be used to guide further support for reducing the costs of cooperation, e.g., making the decisions about code reviews [72] and pull requests [73] more transparent and accessible.

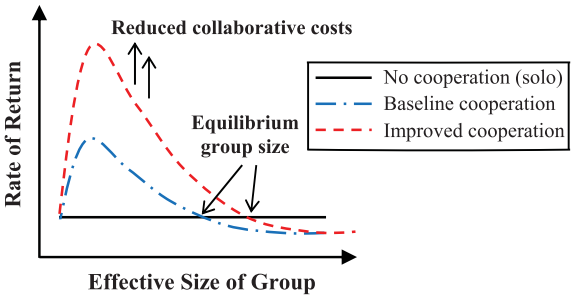


Fig. 9. Reducing the costs of cooperation extend the tail of the rate of returns curve, which also extends the point at which the curve crosses the solo information foraging threshold. Consequently the equilibrium group size is predicted to increase. The figure is adapted from Pirolli's elementary social information foraging model [21].

B. Recommendation Systems

While social coding makes the work more visible [70], it also makes the information overload problem more challenging. To address the challenge, the emerging recommendation systems are ready to become part of software practitioners' toolboxes. These systems are aimed at providing information items estimated to be valuable for a software engineering task in a given context, and are particularly useful in supporting decision making when developers lack experience or cannot consider all the data at hand [26].

Closely related to this paper are approaches recommending *who* in the developer social group, including who should fix an incoming bug [50], who should mentor OSS newcomers [74], who should be working together [56], and who should awareness be attended to [75]. When making these recommendations and the like, *how many* developers to consider can be answered via the group size model that this paper uses. This complements a set of existing recommenders by offering a theoretical underpinning for rationalizing the size of their output.

An assumption of the elementary social information foraging model is that there are a finite number of discoveries to be made in a domain, and once a particular discovery is made it is of no additional value for others to repeat the same discovery [21]. While we believe this holds in general for successfully solving a software change task, special cases may require the assumption to be adjusted. In addition, the assumption, $n = H$, characterizes the case in which members of the group contribute distinctively effective hints [21].

In other group foraging situations, hints may have different weights. For example, the information shared by experts can be estimated to bear more value and such expertise can be identified by the quantification of experience [76], developer's centrality in the social network [45], degree-of-knowledge of source code familiarity [77], or other means. In any case, the updated assumption on n and H can be fed back into the social information foraging model—(2) in particular—for generating recommendations which are better suited to the specific situation.

VI. Conclusion

The main contributions of this paper are the evolutionary-ecological understanding of developers' information foraging in social groups, the theoretical analysis of group size and its relation to individual's rate of gain, the empirical evaluation of a set of hypotheses enabled by the novel perspective, and the concrete insights of applying our research to study developers' rational behaviors in performing a wide variety of information-intensive tasks in software engineering.

Open source projects represent the cleanest way to group developers and other stakeholders together that may or may not be company-specific [71]. Understanding the important structural variable—group size—and its role in OSS development and evolution could well hold useful lessons for how commercial software organizations might be managed. Our future work includes carrying out more empirical studies ideally with proprietary software projects, performing sensitivity analyses of key assumptions and decisions in this paper, and extending the foraging-theoretic analysis to support other activities such as productivity retrospection [48], subgroup discovery [46], and social norm learning [36].

Acknowledgment

The authors would like to thank G. Bradshaw for comments on earlier drafts of this paper.

References

[1] W. Wang and Y. Jiang, "Community-aware task allocation for social networked multiagent systems," *IEEE Trans. Cybern.*, vol. 44, no. 9, pp. 1529–1543, Sep. 2014.

[2] B. Zhu, Z. Xu, and J. Xu, "Deriving a ranking from hesitant fuzzy preference relations under group decision making," *IEEE Trans. Cybern.*, vol. 44, no. 8, pp. 1328–1337, Aug. 2014.

[3] T. P. Pavlic and K. M. Passino, "Distributed and cooperative task processing: Cournot oligopolies on a graph," *IEEE Trans. Cybern.*, vol. 44, no. 6, pp. 774–784, Jun. 2014.

[4] L. D. Xu and W. Viriyasitavat, "A novel architecture for requirement-oriented participation decision in service workflows," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1478–1485, May 2014.

[5] W. He and L. D. Xu, "Integration of distributed enterprise applications: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 1, pp. 35–42, Feb. 2014.

[6] J. D. Blackburn, G. D. Scudder, and L. N. Van Wassenhove, "Improving speed and productivity of software development: A global survey of software developers," *IEEE Trans. Softw. Eng.*, vol. 22, no. 12, pp. 875–885, Dec. 1996.

[7] P. C. Pendharkar and J. A. Rodger, "Probabilistic and analytical estimation of software development team size," *Int. J. Hybrid Intell. Syst.*, vol. 7, no. 2, pp. 137–153, Jun. 2010.

[8] M. Di Penta, M. Harman, G. Antoniol, and F. Qureshi, "The effect of communication overhead on software maintenance project staffing: A search-based approach," in *Proc. Int. Conf. Softw. Maint. (ICSM)*, Paris, France, Oct. 2007, pp. 315–324.

[9] F. P. Brooks, Jr., *The Mythical Man-Month: Essays on Software Engineering*. Reading, MA, USA: Addison-Wesley, 1975.

[10] C. Bird, D. S. Pattison, R. M. D'Souza, V. Filkov, and P. Devanbu, "Latent social structure in open source projects," in *Proc. ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*, Atlanta, GA, USA, Nov. 2008, pp. 24–35.

[11] N. Niu, L. D. Xu, and Z. Bi, "Enterprise information systems architecture—Analysis and evaluation," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2147–2154, Nov. 2013.

[12] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *IEEE Trans. Softw. Eng.*, vol. 32, no. 12, pp. 971–987, Dec. 2006.

[13] P. Pirolli, *Information Foraging Theory: Adaptive Interaction With Information*. New York, NY, USA: Oxford Univ. Press, 2007.

[14] D. W. Stephens and J. R. Krebs, *Foraging Theory*. Princeton, NJ, USA: Princeton Univ. Press, 1987.

[15] J. Lawrance, R. Bellamy, and M. Burnett, "Scents in programs: Does information foraging theory apply to program maintenance?" in *Proc. IEEE Symp. Vis. Lang. Human-Centric Comput. (VL/HCC)*, Coeur d'Alene, ID, USA, Sep. 2007, pp. 15–22.

[16] J. Lawrance *et al.*, "How programmers debug, revisited: An information foraging theory perspective," *IEEE Trans. Softw. Eng.*, vol. 39, no. 2, pp. 197–215, Feb. 2013.

[17] M. T. Su, E. Tempero, J. Hosking, and J. Grundy, "A study of architectural information foraging in software architecture documents," in *Proc. Joint Working IEEE/IFIP Conf. Softw. Archit. (WICSA)*, Helsinki, Finland, Aug. 2012, pp. 141–151.

[18] N. Niu, A. Mahmoud, and G. Bradshaw, "Information foraging as a foundation for code navigation (NIER Track)," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, Honolulu, HI, USA, May 2011, pp. 816–819.

[19] N. Niu, A. Mahmoud, Z. Chen, and G. Bradshaw, "Departures from optimality: Understanding human analyst's information foraging in assisted requirements tracing," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, San Francisco, CA, USA, May 2013, pp. 572–581.

[20] S. D. Fleming *et al.*, "An information foraging theory perspective on tools for debugging, refactoring, and reuse tasks," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 2, Mar. 2013, Art. ID 14.

[21] P. Pirolli, "An elementary social information foraging model," in *Proc. Conf. Human Factors Comput. Syst. (CHI)*, Boston, MA, USA, Apr. 2009, pp. 605–614.

[22] H. Qiao, Y. Li, T. Tang, and P. Wang, "Introducing memory and association mechanism into a biologically inspired visual model," *IEEE Trans. Cybern.*, vol. 44, no. 9, pp. 1485–1496, Sep. 2014.

[23] A. B. Özgüler and A. Yildiz, "Foraging swarms as Nash equilibria of dynamic games," *IEEE Trans. Cybern.*, vol. 44, no. 6, pp. 979–987, Jun. 2014.

[24] E. Nichols, L. McDaid, and N. H. Siddique, "Biologically inspired SNN for robot control," *IEEE Trans. Cybern.*, vol. 43, no. 1, pp. 115–128, Feb. 2013.

[25] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Leveraging transparency," *IEEE Softw.*, vol. 30, no. 1, pp. 37–43, Jan./Feb. 2013.

[26] M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, *Recommendation Systems in Software Engineering*. Berlin, Germany: Springer, 2014.

[27] E. H. Chi *et al.*, "The Bloodhound project: Automating discovery of Web usability issues using the InfoScent simulator," in *Proc. Conf. Human Factors Comput. Syst. (CHI)*, Ft. Lauderdale, FL, USA, Apr. 2003, pp. 505–512.

[28] S. H. Clearwater, T. Hogg, and B. A. Huberman, "Cooperative problem solving," in *Computation: The Micro and Macro View*, B. A. Huberman, Ed. Singapore: World Scientific, 1992, pp. 33–70.

[29] C. W. Clark and M. Mangel, "The evolution advantages of group foraging," *Theor. Popul. Biol.*, vol. 30, no. 1, pp. 45–75, Aug. 1986.

[30] Q. Hong, S. Kim, S. C. Cheung, and C. Bird, "Understanding a developer social network and its evolution," in *Proc. Int. Conf. Softw. Maint. (ICSM)*, Williamsburg, VA, USA, Sep. 2011, pp. 323–332.

[31] A. Kittur, B. Suh, B. A. Pendleton, and E. H. Chi, "He says, she says: Conflict and coordination in Wikipedia," in *Proc. Conf. Human Factors Comput. Syst. (CHI)*, San Jose, CA, USA, Apr./May 2007, pp. 453–462.

[32] L. Williams and A. Cockburn, "Agile software development: It's about feedback and change," *IEEE Comput.*, vol. 36, no. 6, pp. 39–43, Jun. 2003.

[33] W. S. Humphrey, "The team software process," Softw. Eng. Inst., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2000-TR-023, Nov. 2000.
[34] M. A. Cusumano and R. W. Selby, *Microsoft Secrets*. New York, NY, USA: Free Press, 1998.
[35] M. Hoegl, "Smaller teams—Better teamwork: How to keep project teams small," *Bus. Horizons*, vol. 48, no. 3, pp. 209–214, May/Jun. 2005.
[36] C. Yu, M. Zhang, and F. Ren, "Collective learning for the emergence of social norms in networked multiagent systems," *IEEE Trans. Cybern.*, vol. 44, no. 12, pp. 2342–2355, Dec. 2014.
[37] L. H. Putnam, "A general empirical solution to the macro software sizing and estimating problem," *IEEE Trans. Softw. Eng.*, vol. 4, no. 4, pp. 345–361, Jul. 1978.
[38] D. Rodríguez, M. A. Sicilia, E. García, and R. Harrison, "Empirical findings on team size and productivity in software development," *J. Syst. Softw.*, vol. 85, no. 3, pp. 562–570, Mar. 2012.
[39] M. Heričko, A. Živkovič, and I. Rozman, "An approach to optimizing software development team size," *Inf. Process. Lett.*, vol. 108, no. 3, pp. 101–106, Oct. 2008.
[40] P. C. Pendharkar, J. A. Rodger, and G. H. Subramanian, "An empirical study of the Cobb–Douglas production function properties of software development effort," *Inf. Softw. Technol.*, vol. 50, no. 12, pp. 1181–1188, Nov. 2008.
[41] P. C. Pendharkar, G. H. Subramanian, and J. A. Rodger, "A probabilistic model for predicting software development effort," *IEEE Trans. Softw. Eng.*, vol. 31, no. 7, pp. 615–624, Jul. 2005.
[42] T. K. Abdel-Hamid, "The dynamics of software project staffing: A system dynamics based simulation approach," *IEEE Trans. Softw. Eng.*, vol. 15, no. 2, pp. 109–119, Feb. 1989.
[43] G. Antoniol, A. Cimitile, G. A. Di Lucca, and M. Di Penta, "Assessing staffing needs for a software maintenance project through queuing simulation," *IEEE Trans. Softw. Eng.*, vol. 30, no. 1, pp. 43–58, Jan. 2004.
[44] R. K. Smith, J. E. Hale, and A. S. Parrish, "An empirical study using task assignment patterns to improve the accuracy of software effort estimation," *IEEE Trans. Softw. Eng.*, vol. 27, no. 3, pp. 264–271, Mar. 2001.
[45] A. Meneely and L. Williams, "Socio-technical developer networks: Should we trust our measurements?" in *Proc. Int. Conf. Softw. Eng. (ICSE)*, Honolulu, HI, USA, May 2011, pp. 281–290.
[46] J. M. Luna, J. R. Romero, C. Romero, and S. Ventura, "On the use of genetic programming for mining comprehensible rules in subgroup discovery," *IEEE Trans. Cybern.*, vol. 44, no. 12, pp. 2329–2341, Dec. 2014.
[47] K. Petersen, "Measuring and predicting software productivity: A systematic map and review," *Inf. Softw. Technol.*, vol. 53, no. 4, pp. 343–371, Apr. 2011.
[48] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann, "Software developers' perceptions of productivity," in *Proc. ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*, Hong Kong, China, Nov. 2014, pp. 19–29.
[49] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in GitHub: Transparency and collaboration in an open software repository," in *Proc. Conf. Comput. Support. Cooper. Work (CSCW)*, Seattle, WA, USA, Feb. 2012, pp. 1277–1286.
[50] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 1–35, Aug. 2011.
[51] T. G. Cummings, "Self-regulating work groups: A socio-technical synthesis," *Acad. Manage. Rev.*, vol. 3, no. 3, pp. 625–634, Jul. 1978.
[52] T. Wolf, A. Schröter, D. Damian, and T. Nguyen, "Predicting build failures using social network analysis on developer communication," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, Vancouver, BC, Canada, May 2009, pp. 1–11.
[53] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb, "Tesseract: Interactive visual exploration of socio-technical relationships in software development," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, Vancouver, BC, Canada, May 2009, pp. 23–33.
[54] A. Begel, Y. P. Khoo, and T. Zimmermann, "Codebook: Discovering and exploiting relationships in software repositories," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, Cape Town, South Africa, May 2010, pp. 125–134.
[55] D. Surian, D. Lo, and E.-P. Lim, "Mining collaboration patterns from a large developer network," in *Proc. Working Conf. Reverse Eng. (WCRE)*, Beverly, MA, USA, Oct. 2010, pp. 269–273.
[56] D. Surian *et al.*, "Recommending people in developers' collaboration network," in *Proc. Working Conf. Reverse Eng. (WCRE)*, Limerick, Ireland, Oct. 2011, pp. 379–388.
[57] R. Hoda, J. Noble, and S. Marshall, "Self-organizing roles on agile software development teams," *IEEE Trans. Softw. Eng.*, vol. 39, no. 3, pp. 422–444, Mar. 2013.
[58] C. R. B. de Souza, D. F. Redmiles, and P. Dourish, "'Breaking the code,' moving between private and public work in collaborative software development," in *Proc. Conf. Support. Group Work (GROUP)*, Sanibel, FL, USA, Nov. 2003, pp. 105–114.
[59] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity," in *Proc. Int. Symp. Empir. Softw. Eng. Meas. (ESEM)*, Kaiserslautern, Germany, Oct. 2008, pp. 2–11.
[60] W. J. Conover, *Practical Nonparametric Statistics*. New York, NY, USA: Wiley, 1999.
[61] D. A. Tamburri, P. Lago, and H. van Vliet, "Organizational social structures for software engineering," *ACM Comput. Surv.*, vol. 46, no. 1, pp. 42–76, Oct. 2013.
[62] C. Grbich, *Qualitative Data Analysis: An Introduction*. London, U.K.: Sage, 2012.
[63] O. Nierstrasz, "Software evolution as the key to productivity," in *Proc. Int. Workshop Radical Innov. Softw. Syst. Eng. Future (RISSEF)*, Venice, Italy, Oct. 2002, pp. 274–282.
[64] R. D. Putnam, *Bowling Alone: The Collapse and Revival of American Community*. New York, NY, USA: Simon and Schuster, 2000.
[65] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, Dec. 1972.
[66] R. Conradi and B. Westfechtel, "Version models for software configuration management," *ACM Comput. Surv.*, vol. 30, no. 2, pp. 232–282, Jun. 1998.
[67] L.-T. Cheng, C. R. B. de Souza, S. Hupfer, J. F. Patterson, and S. I. Ross, "Building collaboration into IDEs," *ACM Queue*, vol. 1, no. 9, pp. 40–50, Dec. 2003.
[68] A. Sarma, Z. Noroozi, and A. van der Hoek, "Palantir: Raising awareness among configuration management workspaces," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, Portland, OR, USA, May 2003, pp. 444–454.
[69] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin, "Early detection of collaboration conflicts and risks," *IEEE Trans. Softw. Eng.*, vol. 39, no. 10, pp. 1358–1375, Oct. 2013.
[70] R. Cross, S. P. Borgatti, and A. Parker, "Making invisible work visible: Using social network analysis to support strategic collaboration," *California Manag. Rev.*, vol. 44, no. 2, pp. 25–46, 2002.
[71] A. Begel, J. Bosch, and M.-A. Storey, "Social networking meets software development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder," *IEEE Softw.*, vol. 30, no. 1, pp. 52–66, Jan./Feb. 2013.
[72] O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey, "The influence of non-technical factors on code review," in *Proc. Working Conf. Reverse Eng. (WCRE)*, Koblenz, Germany, Oct. 2013, pp. 122–131.
[73] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in GitHub," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, Hyderabad, India, May/Jun. 2014, pp. 356–366.
[74] G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, "Who is going to mentor newcomers in open source projects?" in *Proc. ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*, Cary, NC, USA, Nov. 2012, pp. 1–11.
[75] C. R. B. de Souza and D. F. Redmiles, "The awareness network, to whom should I display my actions? and, whose action should I monitor?" *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 325–340, May/Jun. 2011.
[76] A. Mockus and J. D. Herbsleb, "Expertise browser: A quantitative approach to identifying expertise," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, Orlando, FL, USA, May 2002, pp. 503–512.
[77] T. Fritz, J. Ou, G. C. Murphy, and E. R. Murphy-Hill, "A degree-of-knowledge model to capture source code familiarity," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, Cape Town, South Africa, May 2010, pp. 385–394.

Tanmay Bhowmik (S'13) received the bachelor's degree in computer science and engineering from the National Institute of Technology, Durgapur, India, in 2007, the M.Sc. degree in computer science and the Ph.D. degree from the Department of Computer Science and Engineering, Mississippi State University, Mississippi State, MS, USA, in 2010 and 2015, respectively.

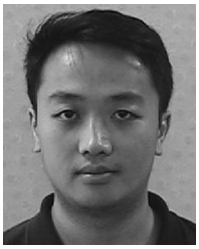
His current research interests include social aspects in software engineering, requirements engineering, software security, big data, and software engineering education.

Mr. Bhowmik is a member of ACM.

Nan Niu (M'08–SM'13) received the B.Eng. degree from the Beijing Institute of Technology, Beijing, China, the M.Sc. degree from the University of Alberta, Edmonton, AB, Canada, and the Ph.D. degree from the University of Toronto, Toronto, ON, Canada, all in computer science.

He is currently an Assistant Professor with the Department of Electrical Engineering and Computing Systems, University of Cincinnati, Cincinnati, OH, USA. His current research interests include software requirements engineering, information seeking in software engineering, and human-centered computing.

Dr. Niu is a recipient of the U.S. National Science Foundation Faculty Early Career Development (CAREER) Award.



Wentao Wang (S'15) received the B.Sc. degree from Shanghai Maritime University, Pudong, China, and the M.Eng. degree from the Beijing Institute of Technology, Beijing, China. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering and Computing Systems, University of Cincinnati, Cincinnati, OH, USA.

His current research interests include software requirements engineering, information seeking in software engineering, and information retrieval.

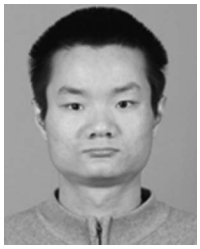
Jing-Ru C. Cheng received the Ph.D. degree in computer science from Pennsylvania State University, State College, PA, USA, in 2002.

She has been a Computer Scientist with the U.S. Army Engineer Research and Development Center, Vicksburg, MS, USA, since 2002. Her current research interests include parallel algorithm development, software tool development for scientific computing, and multiscale multiphysics code development.

Ling Li received the master's and doctorate degrees in production/operations and logistics from the Ohio State University, Columbus, OH, USA, in 1994 and 1996, respectively.

She is a Professor of Production/Operations with Old Dominion University, VA, USA.

Dr. Li has served as an Associate Editor for the IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, and other IEEE journals. She is a fellow in Production and Inventory Management, Association for Operations Management, Chicago, IL, USA.



Xiongfei Cao received the Ph.D. degrees in information systems from the City University of Hong Kong, Hong Kong, and in management science from the University of Science and Technology of China, Hefei, China.

He is an Associate Professor of Management Science and Engineering with the University of Science and Technology of China. His current research interests include knowledge management, IT enabled innovation, and social computing. He has published in referred journals and conference proceedings including *Information Systems Frontier*, *Internet Research*, and the International Conference on Information Systems.