Using Productive Collaboration Bursts to Analyze Open Source Collaboration Effectiveness

Samridhi Choudhary*
Amazon
Alexa Edge ML
Pittsburgh, USA
samridhc@amazon.com

Christopher Bogart
Institute for Software Research
Carnegie Mellon University
Pittsburgh, USA
cbogart@cs.cmu.edu

Carolyn Rose
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, USA
cprose@cs.cmu.edu

Jim Herbsleb
Institute for Software Research
Carnegie Mellon University
Pittsburgh, USA
jdh@cs.cmu.edu

***Abstract*—Developers of open-source software projects tend to collaborate in bursts of activity over a few days at a time, rather than at an even pace. A project might find its productivity suffering if bursts of activity occur when a key person with the right role or right expertise is not available to participate. Open-source projects could benefit from monitoring the way they orchestrate attention among key developers, finding ways to make themselves available to one another when needed. In commercial software development, Sociotechnical Congruence (STC) has been used as a measure to assess whether coordination among developers is sufficient for a given task. However, STC has not previously been successfully applied to open-source projects, in which some industrial assumptions do not apply: management-chosen targets, mandated steady work hours, and top-down task allocation of inputs and targets. In this work we propose an operationalization of STC for open-source software development. We use temporal bursts of activity as a unit of analysis more suited to the natural rhythms of open-source work, as well as open source analogues of other component measures needed for calculating STC. As an illustration, we demonstrate that open-source development on PyPI projects in GitHub is indeed bursty, that activities in the bursts have topical coherence, and we apply our operationalization of STC. We argue that a measure of socio-technical congruence adapted to open source could provide projects with a better way of tracking how effectively they are collaborating when they come together to collaborate.**

***Index Terms*—Socio-Technical Congruence, Productivity, Bursty Collaboration, Hidden Markov Model, Open-Source, GitHub**

I. Introduction

Globally distributed volunteers in open-source projects may only sporadically find themselves available at the same time for tightly-coupled collaboration, since the work rhythms of volunteers with different time zones, work schedules, and levels of commitment will rarely align by chance, in contrast to their brick-and-mortar counterparts. Yet investigations of success in open-source shows that these organizations can still be productive and effective; indeed, bursty patterns of interaction have been shown to be more strongly associated with positive project outcomes than random (Poisson) or periodic time distributions of behavior [1, 2], perhaps because a bursty pattern might reflect people being responsive to each other [3].

Beyond the mere existence of a burst of activity, effective work requires that the right people talk to each other about issues that stand in the way of completing work. This is a concept called Sociotechnical Congruence (STC) [4] – the idea that the communication patterns among a team should reflect the dependency structure among the tasks they are collaborating on. In this work we propose measures of overall burstiness of a project, and of STC adapted to the open-source domain. These measures could be used to monitor and improve processes in medium-to-low intensity open-source projects, in which sporadic collaboration makes it important to make productive use of the little time they have together.

Some research tools such as Tesseract [5], Tukan [6], CollabVS [7], and Palantìr [8] have tried to suggest recommended contacts among developers on this basis. In the field, some recommender bots are used to promote connections between appropriate people. Wessel et al. [9] identified seven bots in use in a sample of 93 projects that helped assign code reviewers, and other bots that might more indirectly draw reviewers into conversation, for example by welcoming newcomers. This published evidence of adoption of such bots demonstrates that these communities are actively interested in fostering the needed coordination and communication.

Although tools, bots, and policies have been adopted in order to improve communication patterns, there has not been a readily available way of assessing the extent to which these measures are working properly. The lack of such measurement technology is even more of a concern when considering that the need for a particular contributor may be implicit and unrecognized, such as in a discussion about a module where an unnamed expert associated with that module does not show up to participate in the discussion. It is currently difficult for projects to maintain awareness of such missed opportunities or their impact on productivity, much less to assess whether such a problem is merely episodic, or is endemic to their typical patterns of coordination. Measuring the adequacy of coordination using a method such as we propose could potentially help not only to test the impact of an introduced bot or policy change, but also perhaps help diagnose needs for volunteers with different patterns of availability.

As STC has previously been measured primarily in commercial projects, in this work we take on two formidable

* Work was done prior to joining Amazon, as a graduate student at CMU.

978-1-7281-5143-4/20/$31.00 © 2020 IEEE

challenges associated with adapting STC to open-source: measuring productivity, and identifying a unit of analysis. First, measuring time to completion of tasks relies heavily on the capability to identify discrete tasks with known endpoints, the set of files associated with each task, developers assigned those tasks, and the communication that occurred for accomplishing the tasks. While these may be known apriori in commercial environments with top-down planning and toolchains, these measures are very difficult to estimate in open-source platforms where "tasks" may not be predefined or assigned. Second, as we will illustrate in this paper, open-source coordination often involves sporadic episodes of relatively intense and explicit collaboration interspersed among periods of individual contributions in which collaboration happens implicitly through the code and artifacts. Measures developed for commercial development by full-time developers may only be valid in open-source during times of active collaborative development. Metrics that straightforwardly incorporate time in such projects may unfairly penalize projects as unproductive for the long spans of time in which participants are not *trying* to work together.

In response to these two challenges, we propose an adaptation of STC that uses empirically observed bursts of collaborative activity as a unit of analysis, and does not rely on the assumption of top-down assignment of tasks, roles, and work schedules to assess team productivity. In particular, we report on an investigation aimed at addressing three research questions in the context of collaboration within more than sixteen thousand PyPI projects on GitHub, investigating each project as an independent collaboration.

Our goal is to adapt the measure of sociotechnical congruence for open-source projects on GitHub, starting by identifying the appropriate time periods as units of analysis during which it is meaningful to measure the productivity of collaboration.

Research Question 1: Are open-source projects' activity traces in GitHub truly "bursty", or is apparent burstiness merely a statistical artifact?

Research Question 2: To what extent do bursts represent meaningful work episodes? More specifically, what operationalization of burst boundaries best mirrors changes in developer task focus? Such boundaries allow us to identify maximally distinct collaboration episodes and isolate them from each other, thus minimizing measurement noise from multiple, topically distinct, adjacent episodes.

Research Question 3: Is the open-source adapted STC measure positively associated with productivity, as the original industrial STC was in its domain?

In the remainder of the paper we first review the literature on identifying bursts in data streams and on coordination in software development. Next, we explain our methodology, outlining the corpus constructed from the PyPI ecosystem on GitHub and the modeling approach that enables us to identify collaborative bursts of activity and define a meaningful measure of productivity. We explain our implementation of the measures of congruence and the related control variables for the collaborative bursts of the projects followed by the results of a quantitative investigation of these bursts. We end by discussing implications of this work and directions for continued research.

II. Related Work

Our approach to studying coordination in open-source projects is twofold. First, we wish to segment the activity timeline of a project into coherent and meaningful 'units' or 'bursts' of activity. Second, we attempt to study coordination and productivity in these projects with the identified bursts functioning as our units of work. We identify related work done on studying and modeling bursty data streams and on the study of coordination in software projects in the following subsections.

A. Bursty Structure in Data Streams

Burstiness can be defined as a violation of the assumptions behind a Poisson process. In a Poisson process, events are distributed randomly in time such that the number of events in two successive intervals is uncorrelated [10]. Time intervals between successive events in a Poisson process follow a Poisson distribution. A process is considered "bursty" if it violates this assumption in the form of a positive correlation between inter-event times, and conversely, long gaps between events are more likely than expected from a Poisson distribution [10, 11]. Similarly, processes with a *negative* correlation between inter-event times are considered periodic, or "anti-bursty". An example of this would be a heartbeat. Bursts have been found in many traces of human activity, such as distributions of letters in text [12], developers' accesses of methods in an IDE [13], or patterns of messages to other individuals in an email archive [14].

Because of the prevalence of burstiness in human activity streams, it makes sense to explore it as a potential lens for understanding open-source work. Rossi et al. [15] attempted to analyze burstiness of open-source projects by separately detecting bursts in three streams of activity: code changes, bug reports, and releases. They found little correlation among them, but their data were aggregated at a monthly level. They speculate that a finer-grained analysis would yield more interesting results. In our work, we follow up by doing a more fine grained analysis, and indeed, the results below report that we find many projects characterized by quite distinct bursts that are 1-5 days in length, much shorter than their window.

In our work, it is important to identify the boundaries of bursts. Different approaches have been applied to measure the *burstiness* of time-series data and identify the boundaries of the bursts themselves. Kleinberg [12] used an infinite state automaton, where each state is a distinctly higher level of activity, to model bursts in document streams over time like email archives. Kleinberg formalizes the bursts for any word as intervals of time that see high frequency of the word occurring in the documents at that time. Lappas et al. [16] used *discrepancy theory* concepts to formalize the notion of burstiness and use it to identify "bursty" intervals for a term

appearing in a document sequence. They use this information to develop a parameter-free, linear-time approach to identify the time intervals of maximum burstiness for a given term.

Bursts make an attractive lens for analysis of open-source work because they appear to be associated with successful collaboration among teams. Riedl and Woolley [1] showed in a controlled study that teams with bursty communication patterns were more successful on a group software development task. Doyle et al. [2] performed simulations of idea-sharing networks, and showed that bursty communication led groups to reach consensus faster.

Many possible distributions are "burstier" than a Poisson process, so there are multiple methods for detecting and characterizing bursts. For example, Howison et al. [3] modeled bursty contribution patterns in Wikipedia as a non-homogeneous Poisson distribution; i.e., a state model switching among Poisson distributions with different parameters for nights/weekends, daytime work hours, and especially high activity periods. Riedl and Woolley [1] used the Fano factor, a simple ratio of standard deviation to mean of inter-event times, to characterize teams as bursty or non-bursty; however, for their analysis they did not try to identify specific burst boundaries. Lappas et al. [16] described a technique called Max-Sum that identifies significant rises and falls in apparent event rates and labels these boundaries as burst boundaries. We explore the applicability of alternative modeling approaches for examining the burstiness of behavior streams in our work and describe our comparison in Sec III-B.

B. Burst Identification Methods

In this research we evaluate three algorithms that identify bursty event streams simply from their timestamp. The maximal sum method identifies periods as long as possible that have more activity than the overall project. Kleinberg assumes there is a cost to increase or decrease in activity of an underlying model, and balances cost of the model with its fit to observed data. HMM also infers an unseen stochastic state model, of which each state probabilistically generates the activity we observe.

1) Maximal Sum Segments (Max-Sum): Lappas et al. [16] proposed a parameter-free, linear time algorithm that we use as our first burst-detection method. They take inspiration from discrepancy theory that is generally used to describe the deviation of a situation from the "expected" behavioral baseline. They define the ***burstiness*** of events in an interval I in a larger sequence S as the difference between the ratio of the *frequency of events* in I and S, and the ratio of *lengths* of I and S:

$$\text{Burstiness} = \left(\frac{\text{frequency in } I}{\text{frequency in } S} - \frac{\text{length of } I}{\text{lengh of } S} \right) \quad (1)$$

Therefore, intervals with frequencies higher than expected have a positive burst score whereas the ones with frequencies lower than expected have a negative burst score. Lappas' algorithm identifies the set of intervals that maximize the length of burstiness functions, using a linear-time 'All Maximal Sum Segments' algorithm [17].

2) Kleinberg Burst Detection: Kleinberg [12]'s method models the structure of bursty document streams over time using an infinite-state automaton, where each automaton state represents a higher level of activity. In the base state q_0, events are assumed to occur at a rate consistent with even distribution throughout the whole dataset's time range. For higher states q_i, the gaps between events are assumed to be shorter by a factor of 2^{-i}. The state transitions are associated with a cost to control the frequency of such transitions to prevent very short bursts due to transient changes in the stream. For a sequence of $n+1$ messages with n *inter-arrival gaps* $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$, the goal is to find a automaton state sequence $\boldsymbol{q} = (q_{i_1}, \cdots, q_{i_n})$, where q_{i_j} best fits the observed data while minimizing number and size of assumed upward state transitions (i.e. a jump from q_0 to q_2 is assumed to be more costly than from q_0 to q_1; transitions from q_1 to q_0 are free).

The HMM formalism reflects the intuition that bursts represent a distinct collaborative phase of activity.

3) Hidden Markov Model: (HMM) A Hidden Markov Model is a doubly stochastic process with an underlying stochastic process that is not observable (it is hidden), but can only be observed through another set of stochastic processes that produce the sequence of observed symbols [18]. It has traditionally been used to characterize the properties of real world signals in a wide range of domains like speech-processing, signal-processing, weather-forecasting and so on.

Elements of an HMM include:

$\boldsymbol{N}$, the number of states S_i in the model. These states are hidden (un-observed) and often have a physical significance. Generally, these states are interconnected in such a way that any state can be reached from any other state.

$\boldsymbol{M}$, the number of distinct observation symbols v_k per state.

$\boldsymbol{A} = \{\boldsymbol{a_{ij}}\}$, a matrix of state transition probabilities of any state to any other state.

$\boldsymbol{B} = \{\boldsymbol{b_j(k)}\}$, a matrix giving the probability of observing a symbol v_k during state S_j.

$\boldsymbol{\pi} = \{\pi_i\}$, the likelihood that the system begins in state S_i.

Given appropriate values of N, M, A, B and π, an HMM can generate an observation sequence $O = O_1, O_2, \cdots, O_T$. The model is trained by adjusting the parameters and the three probability measures A, B and π to maximize $P(O|A, B, \pi)$ on a training set of observation sequences. The model that fits the data best (has the best likelihood for $P(O|A, B, \pi)$) is then selected for further prediction. Given this trained model, it can then be used to compute the optimal state sequence $Q = q_1, q_2, \cdots, q_T$ for any observation sequence $O = O_1, O_2, \cdots, O_T$.

An HMM is more general than Kleinberg's automata model. Unlike the Kleinberg model, an HMM's states are not constrained to have increasing rate of emissions. Instead, the rate of emissions and the probability of transitioning between the states is learned from the observable properties of the data to create a probabilistic model that infers a best-fitting sequence

of states for a data stream. In comparison to the Max-Sum model which is a parameter free model, the HMM has far more parameters to learn. It is more interpretable in the sense that inferences about the states of a trained HMM can be drawn by looking at the distribution of the mean values of the parameters in the states.

4) Topical Segmentation: **TextTiling** [19] is a well-established method of text segmentation designed to segment texts into coherent units that reflect the subtopical structure. It does not identify temporal bursts, but it can nonetheless be used to segment a data stream if the stream contains text. The underlying assumption in this model is that a shift in the word distribution from one segment to the next indicates a shift in the underlying topic of the text. We omit a detailed explanation of the original algorithm, but describe our application of it to software engineering data traces in the methods section.

C. Coordination in Open-Source

Many successful open-source software (OSS) projects are characterized by a globally distributed developer force and a rapid software development process. They succeed despite spanning geographical, organizational and social boundaries, and achieve productive collaboration among their developers. Coordination in these environments can happen both explicitly and implicitly.

Bolici et al. [20] observe that a majority of coordination visible from the behavior traces on open-source projects is *stigmergic*: contributors are seen to coordinate implicitly through changes to the artifact as much or more than they are seen to coordinate explicitly through discussion. Discussion in Github issue traces is generally sparse [21], but explicit coordination also happens in discussions across a profusion of different channels including GitHub, mailing lists, blogs, conferences, personal emails, Slack and Twitter [22]. Dabbish et al. [23] find that software developers generally engage in discussion in order to make social connections with the team when joining an OSS project. Ko and Chilana [24] studied how developers discuss and negotiate problems and solution designs in bug reports. Tsay et al. [25, 26] identified technical and social signals, some of which were related to discussion, that exhibited strong associations with contribution acceptance. Burke and Kraut [27] found that the content of discussions impacted future participation: more polite opening threads were more likely to get a reply. So discussion is clearly a consequential part of coordination in Github projects.

There are multiple challenges associated with appropriately measuring the effectiveness of open-source coordination. One of the difficulties is identifying an appropriate unit of analysis. Rossi et al. [15] noted the bursty nature of open-source projects. However, as far as we know, no research has attempted to empirically investigate these bursts as possible episodes of collaboration. Another difficulty is defining an effective measure of success for the projects. Ghapanchi et al. [28] point out that there is no universally agreed upon definition of success in OSS and there are several factors that contribute to a project's success or failure. They identify six broad areas of success namely: project activity, efficiency, effectiveness, performance, user interest and product quality. Automatic quantification of these measures for GitHub projects has been minimal. Furthermore, there is a very limited understanding of the effect of coordination on these measures.

Prior work has studied the effect of coordination on the productivity of software projects [29]. In particular, Cataldo et al. [4] introduced the measure of *congruence* to quantify the quality of coordination and studied its effect on the effectiveness of commercial software projects. Bolici et al. [21] attempted to explore the effects of congruence in free/libre open-source software (FLOSS) development in GitHub. However, they found that analyzing a project as a whole yielded too little developer communication to assess congruence.

Owing to the diverse and complex nature of OSS, we cannot rely on congruence alone to provide a holistic view of open-source coordination, even along the single dimension of productivity. For example, if the success of a project is measured by the time taken to resolve the issues, there are various external factors that can delay the resolution time, even though the coordination requirements were met by the developers involved. The reasons can range from pending higher priority work, external dependency, negligence on behalf of a developer and so on. These events are not inherently captured by congruence. Investigating them further may help us understand whether these factors are measurable and likely to have an effect. Our eventual aim is to formalize and quantify these factors to characterize collaborative behavior in projects, in order to provide useful information about the project's current, and likely future, challenges. In this paper we lay the foundation by first developing a baseline measure of productive collaboration, so that other hindrances to productivity can be explored in future work.

D. Sociotechnical Congruence

In a software project, task dependencies drive the need to coordinate work activities. One of the important questions about such coordination is who must coordinate with whom in order to get the work done. STC attempts to quantify this coordination. It is the measure of "fit" or "match" between the coordination requirements and the actual coordination activities of a project. In other words, it gives a measure of how much of the coordination that was required to happen, did happen.

Cataldo et al. [4] identified three main components to estimating STC: the dependencies among the tasks (*Task Dependencies*), the people responsible for these tasks (*Task Assignments*) and the actual coordination that occurs among the people (*Actual Coordination*). Congruence is defined as the ratio of the actual coordination to the required coordination. These are further detailed as follows:

Coordination Required (C_R): Given a set of dependencies among the tasks of a project, this people-by-people matrix identifies the individuals that need to coordinate. It is computed using the following matrices:

- *Task Assignment* ($\boldsymbol{T_A}$): a people-by-task binary matrix where $T_A[i][j] = 1$ indicates that $person_i$ is assigned to $task_j$.
- *Task Dependency* ($\boldsymbol{T_D}$): a task-by-task matrix where $T_D[i][j] \neq 0$ indicates that $task_i$ is dependent on $task_j$.
- *Coordination Requirements* ($\boldsymbol{C_R}$): a person-by-person matrix calculated as shown in equation 2 where $C_R[i][j] \neq 0$ indicates that $person_i$ should coordinate with $person_j$ while working on the assigned tasks.

$$C_R = T_A * T_D * T_A{}' \tag{2}$$

Actual Coordination ($\boldsymbol{C_A}$): a people-by-people matrix. $C_A[i][j] \neq 0$ indicates that $person_i$ actually coordinated with $person_j$.

Congruence is then calculated as a logical conjunction between the corresponding cells in the $\boldsymbol{C_R}$ and $\boldsymbol{C_A}$ matrices as shown in equation 3.

$$Congruence(C_R, C_A) = \frac{\sum(C_A \wedge C_R)}{\sum(C_R)} \tag{3}$$

III. Methodology

In this section we explain the protocol used to collect and filter the projects, as well as our implementation and comparison methods for evaluating three algorithms for identifying burst boundaries. Finally, we explain our operationalization of the measures of sociotechnical congruence and associated control measures that are used to examine the effect of coordination on the productivity of a project.

A. Data Collection

We used GitHub's API to collect all commits, issue comments, pull request comments, and open, close, and merge events for 16,337 projects associated with PyPI modules. PyPI[1] is a software ecosystem with a history of fifteen years and a large pool of projects available on GitHub. We chose this domain in order to investigate a popular and relevant type of project while still limiting variability by sticking to a single language and project type. We extracted data from all PyPI modules as of 2016, with GitHub projects having at least 10 stars and 3 contributors, in order to sift out small projects with minimal collaboration needs. We anticipated that collaboration metrics would rarely be meaningful with less than 3 contributors, and since about 90% of GitHub projects have fewer than 10 stars, this limit is a way of choosing a tractable data set of projects that are rated as important by an admittedly rough measure. We have made the data [30] and code [31] available.

B. RQ1: Data Burstiness

One useful and common way of segmenting a GitHub's project trace for research is by *issues* (bug reports or feature requests) or *pull requests* (proposed code changes), since these are identifiable tasks, over an identifiable period, with a known list of participants. However, if development is truly bursty, this unit of analysis could be misleading as a way to study coordination among developers. Comparing two developers' contribution frequency in a long-open issue, for example, would provide little information about their coordination if those contributions were actually concentrated in different bursts months apart. Additionally, projects do not always link issues to the pull requests or other merged code that solves them, so it is difficult to reliably collect all events associated with an issue and its resolution.

[1]https://pypi.python.org/pypi

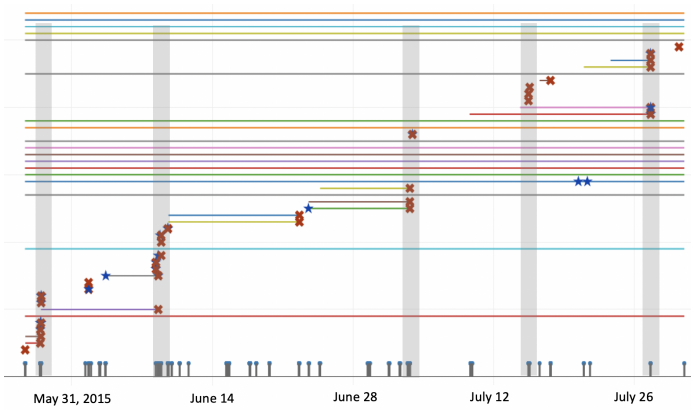

Fig. 1. Bursts cut across issues. In this depiction of activity in a GitHub project, horizontal lines are open issues or pull requests, marked by symbols at the start and end, and vertical grey bars highlight bursts. Short vertical lines at the bottom are commits not associated with an issue.

For those reasons we instead sought to analyze productivity from the perspective of the bursts that seem to emerge from volunteers working when they are motivated, available, and become aware that their help is needed [3]. We first ask whether the bursts really exist as temporal patterns (RQ1), whether they start and end along with one or more topics of collaboration (RQ2), and whether they are an appropriate use as a unit of analysis for measuring STC in open-source projects. (RQ3).

"Burstiness" is the intermittent increases and decreases in activity or frequency of an event. A common measure of burstiness that takes into account temporal dependence in traffic [32] is the ***Fano Factor***, also known as the ***Index of Dispersion***. It is the ratio between the variance and the mean of the event counts, where a value greater greater than 1 signifies a bursty event stream and a value less than 1 depicts a non-bursty stream.

As a first step towards understanding the nature of our activity streams, we computed the *Fano Factor* values for all the projects over their activity streams.

C. RQ2: Burst Detection Models

As Fig. 1 shows, bursts in GitHub activity are often evident in a graph of events over time. They look like flurries of activity separated by quieter periods. However, defining their boundaries can be a difficult judgment call. Recent research [3] has suggested that online collaborative bursts may be caused in part by people being aware of each other's contributions, and responding to them. Such cascades are probably not

the sole driver of bursts, since they could overlap, or fizzle out, or lead participants to notice other unrelated tasks to work on, but we sought to find a segmentation that would approximate these coordinative cascades as well as possible. Therefore, we experimented with three different approaches to delimit the bursts in the project activity timeline, described in Section II-B – a linear time maximal-sum segments algorithm [16], the burst detection method proposed by Kleinberg [12] and a Hidden Markov Model (HMM) [33]. Our goal was to select from this set the approach that yielded what could be considered the most coherent segments of work, i.e., ones that showed greater topic coherence in the discussions within a burst than across those bursts. In order to do this comparison, we used a fourth segmentation method called Text Tiling [19], frequently used in the field of computational linguistics as a means for topic segmentation of running discourse.

The comparison of topical coherence among the first three burst segmentation methods is important, because although a sociotechnical congruence measure should give a high congruence score when distinct groups of people are *simultaneously* working on distinct tasks, and making congruent social connections within those groups, our choice of burst as a unit of analysis means that *shifts of task over time* within a burst will blur the relationship between one collaboration and the next. For example, suppose a developer works on Alice's module A in week 1 of a two-week burst, then works on Bertram's module B in week 2; but doesn't converse with Alice until week 2. A congruence measure applied over the two weeks as a single unit will be unable to detect that the developer failed to talk to Alice in week 1. However, if applied to each week separately, the measure will reflect this failure. For that reason we use topical coherence to assess how well these bursts isolate such shifts of task from each other, in order to get better precision from our congruence measure.

1) Maximal Sum Segments (Max-Sum): For the Maximal Sum Segmentation, we sum up the activity counts at each day in the project timeline and compute the burst scores, and use the maximal sum segments algorithm to identify spans of days with maximal scores. Figure 2 shows the burstiness or the sequence of burst scores for the activity stream of a typical project in our dataset. The activity is bursty in nature with positive (upward facing) bars interspersed with negative (downward facing) bars. A contiguous interval of positive bars with maximal sum forms an activity burst.

2) Kleinberg Burst Detection: We used the *kleinberg()* function in the R "bursts" package for the calculation [34], using total activity during each day of a project as the frequency. Inferred states were then grouped together to define bursts of days with identical states. Therefore, the output is a series of bursts with varying intensities, depending on the automata state assigned to the days in the burst.

3) Hidden Markov Model (HMM): We use a Multivariate Gaussian HMM, a variant where the observation symbols are a combination of multiple gaussian variables: the daily counts of commits, merges, pushes, issue comments, issues opened, issues closed, commit comments, pull request rejections and issues remaining open. We ran the prediction with several choices of N (the number of states), and chose the model with the best fit. The states with mean activity values less than a threshold (we chose a natural cutoff point by visual inspection of activity histograms, which amounted to 3.3 events per day) are labelled as *"dormant"* states and the rest are labelled as *"active"* states. The trained model is used to predict the state sequences for each project. A contiguous run of 'active days' (allowing a maximum gap of 3 days between subsequent active days), is grouped together to define a burst.

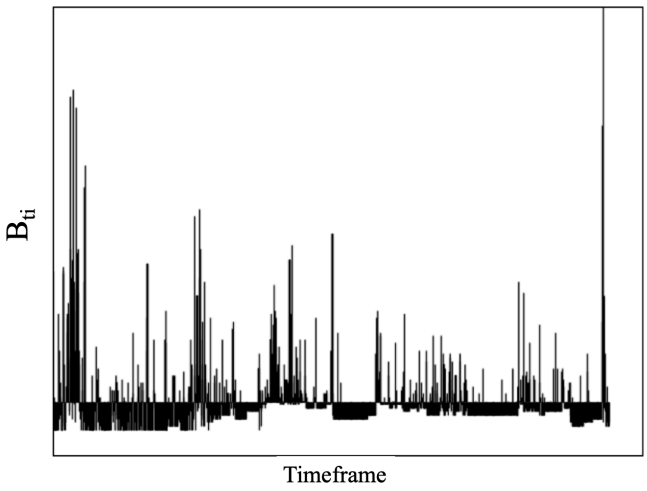

Fig. 2. Burstiness sequence, $\boldsymbol{B}$, as used by the MaxSum algorithm, over GitHub activity counts of a project.

D. Evaluation of Segmentation Methods

To get the cleanest possible signal of activity for Sociotechnical Congruence, we sought segment boundaries that would separate periods where the focus of work is noticeably different. As STC captures the extent to which people are available to one another when they are needed, it would have less utility if the notion of who is needed did not change between segments. If who is needed is related to expertise, then the notion of focus of work should also be related to expertise. We adopt topical content of issue threads as a proxy for relevant expertise. We do not claim that bursts need to be about a single topic for an STC measurement to make sense, merely that there should be some distinction in topic focus across segments of work.

To identify the best performing segmentation method of the three discussed above for our data, we evaluated the topical coherence of the natural language snippets, in the form of issue thread and pull-request discussions, belonging to each burst in the project trace. To do this we applied TextTiling [19] (see Section II-B) to the project history as if it were a long document, and compared the average number of segment boundaries within bursts identified by each of the three burst-detection algorithms to find which models' burst boundaries aligned best with shifts in topic in the events'

associated text. The approach is further elucidated in the following subsections.

Although we use topic segmentation as a means for validating our selection of burst segmentation algorithms, note that it would not have made sense as a primary burst segmentation approach, since discussion is relatively sparse across the timeline of a project. Thus, there are long periods of time during which there would be no signal of whether there is a boundary or not. Furthermore, there is no reason to assume that a single work focus should not have multiple topical segments. Thus, while we expect to see a shift in topic when the burst segment changes, it is not unusual to observe some shifts between topics within a collaborative work episode.

1) Topical Segmentation: To perform topical segmentation we adapted the TextTiling algorithm Hearst [19]: we first *tokenized* the text, removing punctuation and short grammatical words (such as "of", "and", and "the"), and isolated word stems (i.e. we substituted "run" for "running" or "ran") of each comment, commit message, issue title, or pull request title, and computed a word vector for each whole calendar day of activity. We then computed cosine similarity between adjacent days' vectors. Finally we computed a *depth score* for each pair: the sum of the relative heights of the nearest local maximum in each direction; that is, the depth of the bottom of each "trough" in the similarity score graph. We declare a text tile boundary wherever the depth score was greater than $\mu - \frac{\sigma}{2}$, where μ and σ are the mean and the standard deviation of the depth scores for that project. explains the algorithm in more detail.

The segments thus obtained are called 'TT Segments' in our analysis.

2) Model Comparison and Selection (P_k): To compare segmentations, we use Beeferman's P_k [35], a metric of similarity between two sets of segmentation boundaries. It is the probability that "a pair of contributions at a distance of k contributions apart are inconsistently classified; that is, for one of the segmentation outputs the pair lies in the same segment, while for the other it spans a segment boundary." [35] Lower P_k values indicate a higher overlap and are therefore preferred. The ideal value of k, as suggested in their paper, is half the average segment length (in days for our dataset).

For our data, the segmentation boundaries predicted by each of segmentation methods (Max-Sum, Kleinberg, HMM and the evaluation segmentation TT), are projected onto the project timeline. Pairwise P_k values are computed for each pair of segmentations by checking all pairs of days that are k days apart. Since the value of k depends on which segmentation method is selected as the first/reference method (it is the average reference segment length), for each pair we compute the P_k in each direction and take the average value. For example, for HMM and Max-Sum we compute two P_k values: $P_{k_1}(ref = HMM, hyp = MaxSum)$ and $P_{k_2}(ref = MaxSum, hyp = HMM)$. The final value is the average of P_{k_1} and P_{k_2}.

E. RQ3: Coordination - Socio-Technical Congruence

Our final research question employs these HMM-identified activity bursts as a unit of analysis to operationalize a measure of socio-technical congruence [36] and test it is associated with productivity as has been observed in industry. The following sections describe our implementation and the results obtained.

We propose a measure of STC appropriate to the different conditions in open-source, using communication through issue and pull request interactions as the medium of communication. The basic definition of the matrices, explained in Section II-D, remains the same, but we operationalize them differently for GitHub.

Task Assignment $\boldsymbol{T_A[i][j]} \rightarrow$ Since the "task" a person undertakes during a burst may not be clearly defined, we operationalize it as the set of files they changed during the burst. A non-zero value in the matrix at $[i][j]$ indicates that $file_j$ was committed by $person_i$ in the burst.

Task Dependence $\boldsymbol{T_D[i][j]} \rightarrow$ We assume, as Cataldo et al. [36] did, that if files have ever previously been changed together in the same commit, that there is at least an implicit dependency between them, and this implies a dependency among our operationalized tasks. We thus define $T_D[i][j] = 1$ if $file_i$ and $file_j$ have been changed together in some prior commit.

Actual Communication $\boldsymbol{C_A[i][j]} \rightarrow$ In order to communicate with each other for the changes made for an issue or a pull-request (PR) on GitHub, people comment on the discussion thread for those issues and PRs. Therefore, a non-zero value for $C_A[i][j]$ indicates that $person_i$ and $person_j$ communicated with each other by commenting on the same issue thread for the active issues in the burst.

F. Control Measures

Cataldo et al. [4, 36] identified numerous factors that impact the productivity (measured as the resolution time of Modification Requests (MRs)) of the commercial software systems they studied, and used these factors as control measures in their regression models. Since these factors were designed with commercial systems in mind, we cannot directly translate all of them to the open-source systems, and we instead propose an analogous set of factors.

1) **Component Experience** - This measure is calculated as the average number of times that the active committers of the burst had committed the same files prior to the burst. Therefore, for k people (p_1 to p_k) committing a total of n files (f_1 to f_n) in the burst, the component experience is calculated as shown in equation 4 where c_{ij} represents the number of commits made by $person_j$ to the $file_i$ before the current burst started. We applied a log transformation to this variable, since it was heavily skewed towards zero.

$$component\_experience = \frac{\sum_{j=p_1}^{p_k} \sum_{i=f_1}^{f_n} (c_{ij})}{(k * n)} \quad (4)$$

2) **Tenure** - For the active committers in the burst, this measure represents the average number of days that they

had been a part of the GitHub project at the time of completion of that burst. We applied a log transformation since this variable was heavily skewed towards zero.

3) **Number of Issues** - The number of distinct issues or pull requests that were either opened or resolved during the burst. We applied a log transformation since this variable was heavily skewed towards zero.
4) **Number of Teams** - Using the co-commit matrix T_D, we partitioned the graph of co-commits using the fast unfolding method by Blondel et al. [37], in order to infer logical groups of files often changed together. This measure is a count of the distinct number of these file groups involved in the burst.

Since we are interested in capturing productivity of a group of collaborators, we normalize for the size of the group and the length of time by adding them as factors as well:

1) **Committing Users** - The number of people who committed during the burst. We applied a log transformation since this variable was heavily skewed towards zero.
2) **Burst duration days** - The length of the burst in days.

G. Impact on Productivity

In the commercial setting, the goal of a development effort was fixed, but the time was uncertain, so Cataldo used time to completion as a measure of productivity. In the open-source case, the opposite is true: the length of a burst is known, but we have little information about the goal to be achieved. Instead we use lines of code added during the burst as a rough operationalization of productivity of the burst. Both measures can be seen as capturing the ratio of work to unit time, but holding a different part of the ratio fixed. We used code insertions, rather than subtracting removed lines to yield net lines of code, on the theory that replacement of code with no net gain is nonetheless productive. We applied a log transformation since this variable was heavily skewed towards zero. We use a linear regression model detailed in equation 5 in order to understand the impact of congruence and the control measures on the prediction variable - *lines of code added* (log transformed)).

$$log(insertions + 1) = \alpha * (congruence\_measure) + \sum_j \beta_j * (control\_measure)_j + \varepsilon \quad (5)$$

We estimated two models: one without the congruence measure (Model I) and one with the congruence measure (Model II). We discarded bursts where these factors could not be calculated (no commits were made, only one person was present, or no prior file history was available to calculate tenure, experience, modularity, and coordination requirements). The remaining dataset consisted of 45981 bursts across 6401 projects. The parameter estimates for both the models are detailed in the next section.

TABLE I
EFFECTS ON LINES OF CODE ADDED DURING A BURST

	Model I	**Model II**
(Intercept)	0.49^{**}	-0.62^{**}
log(Issue count+1)	1.31^{**}	1.15^{**}
log(Tenure+1)	-0.03^{**}	-0.03^{**}
log(Component Experience + 1)	0.02^{**}	-0.004^{**}
log(Committing Users + 1)	0.30^{**}	0.84^{**}
# Teams	0.22^{**}	0.18^{**}
Burst Duration (days)	-0.003^{**}	-0.004^{**}
Congruence		1.33^{**}
N	45981	45981
Adjusted R^2	0.253	0.288

($^{**}p < 0.0001$, $^{*}p < 0.001$)

IV. RESULTS

A. RQ1: Burstiness is Real

As explained in section III-A, we computed Fano factor over counts of pull request events (open, merge, close, commit, comment) issue events (open, close, and comment) and commits for each project. Measured this way, 99.4% of the 16,337 projects were classified as bursty, that is they had a Fano factor greater than 1.0, with an average of 12.7, and a median of 9.0.

Note that this measure is conservative in that it includes most of the events that are visible through GitHub's affordances: not just actions like issue comments that are explicitly collaborative, but even actions like commits that happen on an individual's own development machine.

B. RQ2: HMM Segmentation Best Aligns with Topic Changes

As Table II shows, HMM-Kleinberg and MaxSum-Kleinberg have a high P_k indicating that the segmentation boundaries drawn by them are quite different, but the low value for HMM and MaxSum suggest that they have similar segment boundaries.

To find the best segmentation method among these three, we computed the pairwise P_k values of each of the methods with the Text Tiling (TT) segments (Table III) as well as the average number of TT segments in one segment of the segmentation method. Kleinberg has the worst P_k values and an average of 7 TT segments per Klein burst. This indicates that one Klein segment contains on average 7 lexically different segments. Between HMM and Max-Sum, HMM has a slightly lower P_k but is much closer to one TT segment per burst than MaxSum. We conclude that our HMM identifies bursts that are more lexically coherent than the units predicted by MaxSum.

The three result in very different boundaries, and of the three, the HMM's boundaries are the most lexically coherent. This comports with HMM's underlying assumption that the data reflect a hidden underlying state, but that particular levels of activity are not best modeled by emphasizing inertia (*contra* Kleinberg), and that there is no apriori reason to insist that burst length should be maximized (*contra* Lappas).

TABLE II
PAIR WISE P_k VALUES. LOWER P_k MEANS SEGMENTATIONS ARE MORE ALIGNED.

Model 1	Model 2	Average P_k
HMM	Klein-Aligned	0.51
HMM	Klein-Raw	0.52
HMM	Max-Sum	0.20
Max-Sum	Klein-Aligned	0.52
Max-Sum	Klein-Raw	0.53

TABLE III
P_k VALUES WITH TEXT-TILED SEGMENTS. LOWER P_k MEANS SEGMENTATIONS ARE MORE ALIGNED.

Model 1	Avg. Segment Length (days)	Avg # TT Segments Per Burst	Avg P_k with TT segments
HMM	39	1.3	0.35
Max-Sum	84	1.8	0.38
Klein	364	7.2	0.41

C. RQ3: Sociotechnical Congruence Associates with Productivity in Bursts

The results of the regression support the applicability of the adapted congruence measure to the GitHub data. Table I shows the results of our regression experiments. Model I is a baseline regression model that considers only the control measures.

The effect of burst duration was unexpected, showing a negative effect. This appears to be influenced by a few outliers in the form of very long bursts that last for years. In some of the largest projects in our dataset, like Ansible, there are so many contributors that significant activity is always going on. If we exclude as outliers the 1.4% of bursts longer than 30 days (614 out of 43689), the burst duration becomes a positive, but still small, effect. This reinforces our caveat that measuring congruence over bursts works best when bursts are defined in a way that they do not encompass many changes of topic.

Number of Teams had a positive effect on productivity. It is a linear factor, so adding a team meant a 22% increase in lines of code added. The 1.3 estimate for *log(issue count)* means that doubling the number of issues increases the lines of code added by 130%, i.e. more than doubling. Recall that we operationalized teams in terms of code files that were worked on together: in other words modularity as revealed by working patterns. Modular code allows teams to work independently with less coordination needed.

Committing Users: Doubling the people involved in a burst means adding 30% more lines of code; this effect indicates that more people produce more code, but there are diminishing returns on adding more people to a burst.

Tenure and *Component Experience* both had very small effects, tenure negative and experience positive (in Model I). Although both are statistically significant effects in this large sample of bursts, they're not meaningfully large effects.

Model II introduces the measure of *Congruence* in our analysis. It has a statistically significant effect on lines of code produced, and slightly improves the model's predictivity. Previous findings on the effect of congruence in large commercial systems have also shown that high congruence leads to higher productivity, as measured, in their case, by shorter resolution times of tasks ("modification requests" in [4, 29, 36]; see Sec. III-G). Our model revealed a smaller contribution due to congruence than Cataldo (e.g. .09 r^2 improvement in predicting issue resolution time [29, Table 5] vs. our .03 r^2 improvement in predicting lines of code added, Table I). However, like Cataldo's results, the actual effect of congruence is practically significant: the coefficient of 1.33 in the table means that the difference between worst (0.0) and best (1.0) coordination by our metric is associated with a 133% increase in lines of code produced in a burst.

Adding congruence to the model increases the R^2 slightly, and changes the coefficients of several of the factors; however relationships among the coefficients' changes appear to be complex. The most changed factor, number of committing users, has a -0.21 correlation with congruence, suggesting that congruence is more critical for small groups of developers.

Although congruence is a continuous measure between 0 and 1, in more than half of bursts (67%) the congruence is either 0.0 (10229 bursts) or 1.0 (20563 bursts), meaning no or perfect coordination according to our operationalization. This distribution is not surprising since many of the features of bursts (committers, issue threads, files touched, or commits) used to calculate the congruence ratio are integers with small median values and long-tailed distributions. Thus it is likely that there are only two of some critical feature, making 0 or 1 the only possible values. For example, the median number of participants in a burst is 4 participants and 21% of bursts have only two participants. As a result, in many cases congruence is simply present or absent. This explains why adding congruence to the model increases the weight of the "committing users" parameter

Note that the relationship we have shown between high STC and lines of code added may not be causative – it could be, for example, that bigger changes attract the appropriate contributors attention. Nonetheless, this shows that congruence is related to this measure of productivity.

V. DISCUSSION

The method we have described produces a congruence ratio for each burst of coordination in a project. This number in itself could be useful for tracking the effects of changes in tools or practices, for example when introducing a bot that does reviewing assignments. However, although the effect of this metric is large in terms of lines of code, it is also noisy – compare our open source model's $R^2 = .29$) with Cataldo's measure [4] for a commercial development setting (.87), which we believe is more easily predicted, for the reasons we described in the introduction.

However, our method also provides incidental information that could be useful for providing more actionable feedback about practices, tools, and individual volunteers.

First, the burst segments themselves are a byproduct of the calculation that project participants could try to align with known events such as scheduled releases, meetings, or hackathons as another way to evaluate how these events contribute to project productivity and engagement.

The matrices underlying the congruence calculation could inform project leadership exactly what module was edited during a burst for which no expert showed up during a discussion about it. Anecdotally, we have observed that sometimes key people not showing up when needed can delay development. A quick check in the dataset of PyPI projects illustrates the problem: when a user is specifically mentioned in an issue comment with an @-sign (i.e. *@-mentions*, a common practice because the mentioned person will receive a notification that their participation is wanted), about 18% of the time the mentioned user does not respond until more than 2 days later – longer than the median length of a burst of activity – thus missing the chance to provide timely help during a productive spurt of collaboration. Prior work by Kavaler et al. [38] showed that such @-mentions were associated with the visibility and productivity of a mentionee, but not with the person's responsiveness, suggesting that the timeliness or responsiveness of contributors is not a particularly visible dimension of their participation. Burst-based STC measures and matrices could help reveal the responsiveness of contributors, as well as reveal contributions patterns of people whose timely contributions have not been explicitly noticed enough to draw @-mentions.

Finally, although our HMM technique produces a fairly rich modeling of project state, inferring a state model based on daily counts of each event type, for segmentation purposes we simply combined the most active states. These states, however, are characterized by different mixes of event types. We expect that more can be learned about the anatomy of bursts by investigating the sequence of such states within bursts, and the frequency of the different states in different kinds of projects. For example, there could be distinct phases of recruitment and retention of developers in a project, or there might exist diverse patterns of work on different artifacts (code, documentation, tests) over a project's lifecycle.

VI. Threats to Validity

Construct Validity For all the volume of data available about open source projects, a lot of the context of development has to be inferred. Lines of code, for example, is known to be an imperfect measure for productivity, however, it is a commonly-used operationalization in the absence of better measures.

Our operationalizations of quantity of activity and connectivity among participants are also likely to be incomplete. Some of the periods we have classified as "low activity" due to a lack of development events in GitHub may in fact involve collaboration on the many other communication platforms that are known to be used by open-source projects [22]. Future work could establish whether activity on other platforms tends to alternate with GitHub activity or correlate with it; however, we hypothesize that in small to medium projects that use GitHub's issue and code review features, GitHub activity will play a key, central role, since its conversations are persistent and tightly linked with the code.

Internal Validity and Conclusion Validity It is difficult to separate cause and effect among the factors associated with bursts: for example poor coordination could cause bursts to end sooner because a needed person was absent, or, conversely, bursts that end sooner could give less time for a needed person to find time to participate. The measurement of productivity itself could as easily be treated an independent or dependent variable. We mitigate this by not describing relationships as causative except when context makes such a relationship clear.

External Validity Finally, we focused on one type of project, PyPI modules, to reduce variability due to differences in language and development style, but this limits our knowledge of how the technique generalizes to other kinds of projects. We have also shown that the burst model does not generalize well to large, busy communities in which development never pauses.

VII. Conclusion

Understanding coordination in open-source projects is not a trivial task. We attempted to study this by adapting a measure of coordination previously used to study commercial software systems, to open-source projects on GitHub, taking into account the bursty, undirected nature of volunteer work. This is a novel effort and one of the few attempts in this domain. We used a Hidden Markov Model to construct a meaningful unit of work and identify coherent phases of activity in a project where coordination might be necessary.

Our results show that coordination, as measured through congruence, is associated with an effect on the productivity of a project in the active phases of its timeline.

This paper presented our effort to synthesize a model of productive collaboration in open-source projects on GitHub. We examined *when* the collaboration occurs, and *who* should coordinate for successful completion of the tasks. We presented and used an unsupervised computational model, as a novel tool that was able to detect interesting and coherent episodes of activity in the open-source projects' timeline. These episodes were used to quantify a meaningful unit of work and a measure of productivity. Our quantitative analysis showed that socio-technical congruence and related control variables had an impact on the productivity of the active phases of the projects. These improved ways of measuring collaborative episodes in open-source environments may help provide ways to empirically ground future improvements to bots and practices that seek to spur appropriate coordination in open-source projects.

Acknowledgments

This research was funded in part by NSF grants IIS 1546393, ACI-1443068, and 1633083; the Alfred P. Sloan Foundation, and the Google Open Source Program Office.

REFERENCES

[1] C. Riedl and A. Woolley, "Teams vs. Crowds: A Field Test of the Relative Contribution of Incentives, Member Ability, and Collaboration to Crowd-Based Problem Solving Performance," *Academy of Management Discoveries*, p. amd.2015.0097, 2016. [Online]. Available: http://amd.aom.org/lookup/doi/10.5465/amd.2015.0097

[2] C. Doyle, B. K. Szymanski, and G. Korniss, "Effects of communication burstiness on consensus formation and tipping points in social dynamics," *Physical Review E*, vol. 95, no. 062303, pp. 32–37, 2017.

[3] J. Howison, J. F. Olson, and K. M. Carley, "Motivation through visibility in open contribution systems," *Group*, 2011.

[4] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley, "Identification of coordination requirements: implications for the design of collaboration and awareness tools," in *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*. ACM, 2006, pp. 353–362.

[5] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb, "Tesseract: Interactive visual exploration of socio-technical relationships in software development," in *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*. IEEE, 2009, pp. 23–33.

[6] T. Schümmer and J. M. Haake, "Supporting distributed software development by modes of collaboration," in *ECSCW 2001*. Springer, 2001, pp. 79–98.

[7] P. Dewan and R. Hegde, "Semi-synchronous conflict detection and resolution in asynchronous software development," in *ECSCW 2007*. Springer, 2007, pp. 159–178.

[8] A. Sarma, Z. Noroozi, and A. Van Der Hoek, "Palantír: raising awareness among configuration management workspaces," in *Software Engineering, 2003. Proceedings. 25th International Conference on*. IEEE, 2003, pp. 444–454.

[9] M. Wessel, B. M. de Souza, I. Steinmacher, I. S. Wiese, I. Polato, A. P. Chaves, and M. A. Gerosa, "The power of bots: Characterizing and understanding bots in oss projects," *Proc. ACM Hum.-Comput. Interact.*, vol. 2, no. CSCW, pp. 182:1–182:19, Nov. 2018. [Online]. Available: http://doi.acm.org/10.1145/3274451

[10] K. I. Goh and A. L. Barabási, "Burstiness and memory in complex systems," *Exploring the Frontiers of Physics2*, vol. 81, no. 48002, pp. 1–5, 2008.

[11] A.-L. Barabási, "The origin of bursts and heavy tails in human dynamics," *Nature*, vol. 435, pp. 207–211, 2005.

[12] J. Kleinberg, "Bursty and hierarchical structure in streams," *Data Mining and Knowledge Discovery*, vol. 7, no. 4, pp. 373–397, 2003.

[13] I. D. Coman and A. Sillitti, "Automated Identification of Tasks in Development Sessions," *International Conference on Program Comprehension*, pp. 212–217, 2008.

[14] A. Perer, B. Shneiderman, and D. W. Oard, "Using Rhythms of Relationships to Understand E-Mail Archives," *J. American Society for Information Science and Technology (ASIS&T)*, vol. 57, no. 14, pp. 1936–1948, 2006.

[15] B. Rossi, B. Russo, and G. Succi, "Analysis of Open Source Software Development Iterations by Means of Burst Detection Techniques," *Proc. International Conference on Open Source Systems (OSS)*, vol. 299, pp. 83–93, 2009.

[16] T. Lappas, B. Arai, M. Platakis, D. Kotsakos, and D. Gunopulos, "On burstiness-aware search for document sequences," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 477–486.

[17] W. L. Ruzzo and M. Tompa, "A linear time algorithm for finding all maximal scoring subsequences." in *ISMB*, vol. 99, 1999, pp. 234–241.

[18] L. Rabiner and B. Juang, "An introduction to hidden markov models," *ieee assp magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[19] M. A. Hearst, "Multi-paragraph segmentation of expository text," in *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 1994, pp. 9–16.

[20] F. Bolici, J. Howison, and K. Crowston, "Stigmergic coordination in FLOSS development teams: Integrating explicit and implicit mechanisms," *Cognitive Systems Research*, vol. 38, pp. 14–22, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.cogsys.2015.12.003

[21] ——, "Coordination without discussion? socio-technical congruence and stigmergy in free and open source software projects," in *Socio-Technical Congruence Workshop in conj Intl Conf on Software Engineering, Vancouver, Canada*, 2009.
FOSE 2014 Proceedings of the on Future of Software Engineering, pp. 100–116, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2593882.2593887

[23] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*. ACM, 2012, pp. 1277–1286.

[24] A. J. Ko and P. K. Chilana, "Design, discussion, and dissent in open bug reports," in *Proceedings of the 2011 iConference*. ACM, 2011, pp. 106–113.

[25] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *Proceedings of the 36th international conference on Software engineering*. ACM, 2014, pp. 356–366.

[26] ——, "Let's talk about it: evaluating contributions through discussion in github," in *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*. ACM, 2014, pp. 144–154.

[27] M. Burke and R. Kraut, "Mind your ps and qs: the impact of politeness and rudeness in online communities," in *Proceedings of the 2008 ACM conference on Computer supported cooperative work*. ACM, 2008, pp. 281–284.

[28] A. H. Ghapanchi, A. Aurum, and G. Low, "A taxonomy for measuring the success of open source software projects," *First Monday*, vol. 16, no. 8, 2011.

[29] M. Cataldo and J. D. Herbsleb, "Coordination breakdowns and their impact on development productivity and software failures," *IEEE Transactions on Software Engineering*, vol. 39, no. 3, pp. 343–360, 2013.

[30] S. S. Choudhary, C. Bogart, C. P. Rosé, and J. D. Herbsleb, "Modeling productivity in open source github projects: A dataset and codebase," https://doi.org/10.1184/R1/6397013.

[31] S. Choudhary, "Modeling productivity in open source github projects," https://github.com/samridhishree/Github_Productivity, 2019.

[32] V. S. Frost and B. Melamed, "Traffic modeling for telecommunications networks," *IEEE Communications Magazine*, vol. 32, no. 3, pp. 70–81, 1994.

[33] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[34] J. Binder, *bursts: Markov model for bursty behavior in streams*, 2018, r package version 1.0-1. [Online]. Available: https://CRAN.R-project.org/package=bursts

[35] D. Beeferman, A. Berger, and J. Lafferty, "Statistical models for text segmentation," *Machine learning*, vol. 34, no. 1, pp. 177–210, 1999.

[36] 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 *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. ACM, 2008, pp. 2–11.

[37] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[38] D. Kavaler, P. Devanbu, and V. Filkov, "Whom are you going to call? determinants of@-mentions in github discussions," *Empirical Software Engineering*, pp. 1–29, 2018.