Modeling Coordination and Productivity in Open-Source GitHub Projects

Samridhi Shree Choudhary **Christopher Bogart**
Carolyn Penstein Rosé **James D. Herbsleb**

June 2018
CMU-ISR-18-101

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

The work was funded by National Science Foundation Information and Intelligent Systems Division (IIS) under grant numbers 1546393 and 1633083; the Alfred P. Sloan Foundation, and the Google Open Source Program Office.

Keywords: Bursty Streams, Socio-Technical Congruence, Productivity, Activity Phases, Hidden Markov Model, Open-Source, GitHub

Abstract

In open-source software development, coordination between globally separated developers is often structured in ways not immediately visible to them, such as implicit groupings of people working on similar code and related issues. This opacity is despite the availability and accessibility of a large quantity of low-level project activity data on platforms like GitHub. This paper uses this low-level data to construct meaningful indicators that could offer improved transparency into how coordination is conducted in open source. Prior work has shown the value of *Socio-Technical Congruence* for evaluating the quality of coordination in commercial software systems. However, little work has successfully translated this analysis to the domain of open source, primarily due to their less formal and inconsistent ways of partitioning work, assigning tasks and measuring success. We present a technique for distinguishing the active phases of coordination and define a measure of productivity for these projects. We perform a quantitative analysis of the influence of congruence on productivity in these phases demonstrating that the associations between our measure of productivity and the measures of congruence and other control variables are subtle but consistent with the prior work in commercial software development and discuss some applications of our work.

1 Introduction

In open-source ecosystems, decentralized decision-making among highly interdependent yet independently managed projects becomes more and more challenging as their increasing scale makes it difficult to assess the health of a project, either from the outside when choosing to adopt or contribute to it, or from the inside when seeking to maintain a healthy collaboration. With open source moving to a central role in software supply chains, the economic stakes of these decisions are escalating. The ebb and flow of resources in these dynamical systems creates instability and the need for quick and effective decision making.

Popular online work environments like GitHub[1] provide detailed traces of work activities for various open-source projects. These include the substance and structure of software artifacts along with the maintenance metadata, such as code changes, bug reports, code reviews and feature requests. The traces also contain discussion among the developers capturing explicit coordination between them. However, while they are openly available and can be used to support decision making, their large volume can make it noisy and difficult to contextualize; many transparent details become opaque at a large scale. This paper presents a mixed-methods investigation to inform future efforts at increasing visibility into the inner workings of open-source software development on GitHub by constructing meaningful indicators from the existing low-level information.

Recent research has shed light on how developers make important decisions using the vast pool of information available on environments like GitHub [13]. For each decision, there are a number of "hidden qualities" that developers care about, for example, whether a community responds to problems in a timely manner, whether the code is of good quality and whether the developers are open to external contributions. These traits are very difficult or expensive to observe directly. The developers tend to use visible information, emitted intentionally or unintentionally by other developers, as "signals" to infer the presence or absence of these qualities [29]. Many such signals have been identified in qualitative studies, with the strength of the signals estimated in quantitative work [40]. The long term goal of our research is to construct signals related to the important qualities of work in open-source systems that are readily observable and interpretable. In this paper, we focus on a property that represents the extent to which a community is able to solve its problems in a timely manner with effective coordination. We refer to this measure as *productivity* of open-source projects.

Coordination in software development teams has been a recurrent topic of interest in software engineering research. Two of the most popular approaches in this regard are that of ***Modularity*** and ***Congruence***. Modularity is based on a decomposability view of the software systems [6, 20]. The primary aim of a modular design is to reduce technical dependencies among different components of the system leading to greater productivity and faster resolution times. Although this line of research has been quite influential, it assumes a static view of the system and is not sufficient for situations where the product dependencies rapidly change [28, 12]. Congruence offers a different perspective on coordination that takes into account the dynamic nature of dependencies. Cataldo et al. [8] introduced the concept of socio-technical congruence. They demonstrated that organizations are more successful when there is a 'match' between their coordination *requirements*, implied

[1]http://www.github.com

by the structure of the technical dependencies, and their *capability* to coordinate, as measured by the organization's communication patterns [9].

Although these measures have provided a useful insight into the organization of software development, the focus of the research has been on conventional commercial software development. There are two formidable challenges associated with generalizing these measures to open-source platforms and practices. First, calculating congruence relies heavily on the capability to identify the formal structure of the work: discrete tasks, the set of files changed as a part of each task, developers who are assigned those tasks and the communication associated with the tasks. While this data is sometimes available in commercial settings, they are quite difficult to operationalize in open-source platforms in which tasks, teams and responsibilities may be defined less formally, if at all, and the work is self-assigned by volunteers. 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 no contribution or individual contributions with implicit collaboration. Congruence might be quantifiable and valid only for the explicit collaborative episodes.

Specifically, this paper reports on an investigation aimed at addressing the following research questions:

Research Question 1: How can the activity traces of open-source software be structured to identify meaningful units of collaborative work, that would enable investigation of the effect of coordination on the productive accomplishment of that work?

Research Question 2: How can the traditional measures of congruence be adapted for open-source projects on GitHub, and is the effect of these measures on project productivity consistent with that found in commercial projects?

In the remainder of the paper we first review the literature on structuring and segmenting data streams into units of high activity frequency and the prior work on coordination in software development. Next, we explain our mixed-methods approach, beginning with a description of the corpus that we constructed from the PyPi ecosystem on GitHub and the modeling approach used to identify the collaborative phases of activity. With this as a foundation, we define a meaningful measure of productivity. We explain our operationalization of the measures of congruence and the related control variables for the collaborative phases of projects. We end with a discussion of the potential uses of not only the productivity measure to assess the overall effectiveness of a project's collaboration, but also of the components of the model as potential and useful corrective indicators for the project participants.

2 Literature Survey

As evident from the stated research questions, our approach to study coordination in open-source projects is two fold. 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 bursts functioning as our units of work. We review the literature pertaining to work done on studying and modeling bursty data streams and on the study of coordination in software projects in the following subsections.

2.1 Bursty Structure in Data Streams

A plethora of models have been developed to account for the flow of events in data streams over time, many of them referring to the bursty nature of the flow. Burstiness has been modeled in diverse types of streams including that of the words within a stream of documents [24] to events within telecommunication networks [18, 16].

A considerable number of approaches have been proposed for efficient burst-detection [43, 42, 24]. The method proposed by Fung et al. [19] clusters bursty terms together as a representation of what might be thought of as events at different points in time according to the temporal flow of documents. Lappas et al. [27] use *discrepancy theory* concepts to model the burstiness of a term in a sequence of documents over time. They formalize the notion of burstiness by defining a 'burst score' of a term for an interval of time. A parameter-free, linear-time algorithm is then used to identify the time-intervals that maximize the burst score of any given term over the entire period.

A seminal method of burst detection, which has motivated substantial work in this field, was proposed by Kleinberg [24]. His approach uses an infinite-state automaton to model the structure of bursty document streams over time, in which each automaton state represents a different frequency at which a word appears in the text. At any time the automaton can be in one of the underlying set of states that correspond to increasingly rapid rates of emission. The onset of a burst is signaled by a state transition from a lower to a higher state. State transitions are associated with a cost in order to limit the frequency of such transitions.

Kleinberg [24]'s automaton model draws on the formalism of Hidden Markov Models, another popular method for studying patterns in sequential data ranging from real world signals like speech processing [31] to the domains of computational biology [26]. A Hidden Markov Model (HMM) 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. In our current work we compare HMMs with Kleinberg [24]'s and Lappas et al. [27]'s models as different ways to characterize the bursty nature of activity streams for projects on GitHub.

2.2 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 achieving productive collaboration among developers. Coordination in these environments can happen both implicitly and explicitly. Bolici et al. [5] observe that most 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. However, this does not mean that explicit planning and negotiation does not happen in open-source projects. Explicit coordination can happen through discussions across a profusion of different channels including GitHub, mailing lists, blogs, conferences, personal emails, Slack and Twitter [39]. Dabbish et al. [13] 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 [25] studied how developers discuss and negotiate problems and solution designs in bug reports. Tsay et al. [40] found technical and social signals that exhibited strong associations with contribution acceptance for projects on GitHub. Tsay et al. [41] examined how open-source developers discuss and evaluate the pull requests on GitHub. These discussions have important implications for project management. On platforms like GitHub, discussion is primarily visible in issue threads and pull requests. Although prior work has noted that GitHub discussion traces are sparse [4], communication through discussion has nonetheless been found to be consequential.

There are multiple challenges associated with appropriately measuring the effectiveness of coordination in open source. One of the difficulties is identifying an appropriate unit of analysis. Rossi et al. [33] noted the bursty nature of open-source projects. However, as far as we know, there have been few attempts to empirically investigate these bursts as possible units of activity in an open-source environment. A recent work in this field is that of Riedl and Woolley [32] that study collaboration in crowd sourced environments. They performed a large-field experiment on team collaboration on an online platform and found that temporal "burstiness" of team activity and the diversity of information exchanged among team members are strong predictors of performance. More generally, Barabási [1] suggest that human activities ranging from communication to work patterns are found to be inherently bursty indicating that when individuals execute tasks based on some perceived priority, the timing of the tasks will be heavy tailed. In other words, rather than a randomly distributed pattern of communications, there tends to be periods of high activity followed by periods of little to no activity. Olson et al. [30], in contrast, modeled Wikipedia contribution burstiness as a function of participants' awareness of each others' activity, as well as circadian and weekly activity cycles. Another difficulty is defining an effective measure of success for the projects. Ghapanchi et al. [21] 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 proven challenging.

Prior work has studied the effect of coordination on the productivity of software projects [7]. In particular, Cataldo et al. [8] introduced the measure of *congruence* to quantify the quality of coordination and studied its effect on the effectiveness of commercial software projects. However, owing to the diverse and complex nature of OSS and the differing standards of project organization among platforms, they have not been successfully and accurately translated to these domains.

3 Data

In this section, we explain our dataset[11] and the protocol used to collect and filter the projects. We also study the bursty nature of the data and quantify the burstiness of the project activity streams.

3.1 Data Collection

Congruence is a somewhat subtle effect, and previous studies of congruence in commercial systems have detected it by focusing their analysis on the data from a single company at a time, using a limited number of languages and technologies. We wanted to choose a similarly homogeneous open source dataset, in order to be consistent with this characteristic of prior work. We sought a single software ecosystem with enough history and multiple projects on GitHub as our data source, using a single language.

We thus chose the PyPi[2] ecosystem as our domain of choice. It is a software ecosystem with a history of fifteen years and a large pool of projects available on GitHub, written in a language that the researchers were familiar with (Python). We began with a list of 48,668 GitHub repositories culled from the PyPi metadata records. We then filtered for projects having at least 10 stars and 3 contributors in order to sift out small inconsequential projects without collaboration needs, leaving us with 16,683 projects.

We used the GitHub API[3] to scrape issues and pull requests, along with associated comments, commits, close and merge events, as well as commits and commit comments that were not associated with pull requests. We used the githubarchive repository[4] to identify push events, which are not available from the GitHub API. We took data from both sources from the earliest data available (2011) up through December 31, 2016.

3.2 Data Burstiness

In connection to the research question 1 (Section 1), we explore how to segment a stream of work into episodes that can be used to measure productivity. Noting that the events in projects appear to be bursty, we first test how pervasive the bursts are in the data.

In statistics, "burstiness" is defined as the intermittent increases and decreases in activity or frequency of an event. One of the most common and simplistic measures of burstiness 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 than 1 signifies a bursty event stream and a value less than 1 depicts a non-bursty stream. This measure has invariably been applied as a first step to understand the nature of any stream [18]. As mentioned before (section 3.1), we collect several variables related to daily project activities. These include number of commits, merges, pushes, issue comments, issues opened, issues closed, commit comments, pull request rejections and issues remaining open. In order to compute the fano factor for each project, we sum the counts of all the activities of the projects per unit time (day) to get the total activity counts per day. The mean and the variance are computed for the series of daily counts per project to get the project Fano factor. The following were the results of this experiment:

- Average value of the Fano Factor for projects = $\mathbf{8.82}$
- Median value of the Fano Factor for projects = $\mathbf{4.83}$
- Percentage of projects with Fano Factor > 1 = $\mathbf{94.1\%}$

[2]https://pypi.python.org/pypi
[3]https://developer.github.com/v3/
[4]https://www.githubarchive.org/

On average (mean or the median) the fano factor for the projects in our data is greater than 1. Moreover, roughly 94% of the projects have fano factor greater than 1, representing a large majority of the projects having bursty activity streams. With these numbers at hand, we conclude that the project activity data can be characterized as bursty, and we can proceed to identify and delimit the intervals of time with high and low (or no) activity.

4 Methodology

This section elaborates on our approach to understand collaboration and productivity in open-source projects hosted on GitHub. We deploy three segmentation models to identify active bursts or phases of high activity in our data. This is followed by a comparison among the segmentation models in order to choose the most appropriate one. Finally, we explain our operationalization of coordination using the measure of socio-technical congruence. We detail the other control measures that we use along with the congruence values to study the effect of coordination on the productivity of a project.

4.1 Burst Detection Models

Given the evidence of the bursty nature of activity streams in GitHub, and considering the variety of approaches to model the bursts, we compared and contrasted three alternative methods: the linear time maximal-sum segments algorithm [27], the seminal burst detection method proposed by Kleinberg [24] and a Hidden Markov Model (HMM) trained on the daily activity counts of the project [31]. The pairwise alignment of their predicted burst boundaries is measured using Beeferman et al. [2]'s $\boldsymbol{P_k}$, that is often used to evaluate the alignment of segmentation algorithms .

In order to measure cohesion within the segments, we construct a parallel behavior stream using the developer conversations that happen within a project by looking at the text from issue threads, pull requests, and commit comments. Breaks in lexical cohesion are identified by a change in focus (topics) of the verbal interactions to detect places where the project focus has changed. In order to identify such breaks, we use the standard TextTiling approach [22]. The lexical segments thus obtained are compared with the segments obtained from each of the three burst-detection algorithms using $\boldsymbol{P_k}$. The model with the best $\boldsymbol{P_k}$ value with the text segments represents a segmentation that is the closest to the lexically coherent segmentation of the activity and is selected for obtaining the final bursts of activity for a project. The process is detailed in the following subsections.

Maximal Sum Segments *(Max-Sum)* - We use the method suggested by Lappas et al. [27] as our first burst-detection method. The model defines and calculates a burst score associated with different intervals of time and uses a linear-time variation of the 'All Maximal Sum Segments' algorithm [34], to identify the set of intervals that maximize the score. For our data, we use a count of activities for each day in each project. A positive or negative burst score is computed for each day in the project timeline. The algorithm is then used to identify intervals of days with maximal scores that are interpreted as "work episodes" or "activity bursts". Figure 1 shows the sequence of burst scores for the activity stream of a random project in our dataset. The x-axis represents time and the y-axis represents burst score. Bursts appear in the figure as groups of

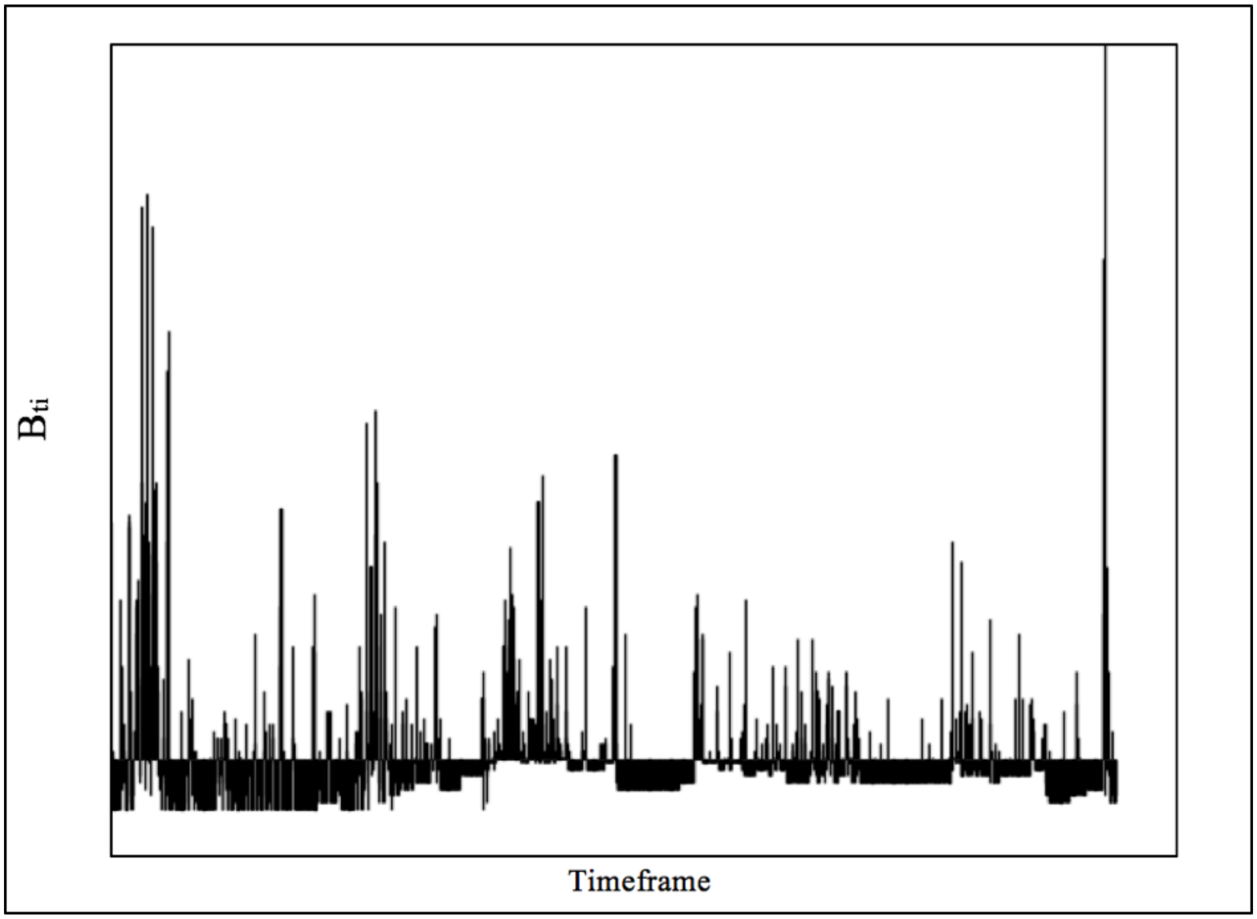

Figure 1: Burstiness sequence, $\boldsymbol{B}$, for the GitHub activity counts of a project. Contiguous intervals of upward-facing bars represent a burst of activity.

positive (upward facing bars) scores interspersed with negative (downward facing bars) scores. A contiguous interval of positive bars with maximal sum forms an activity burst. Graphs similar to this were plotted for many projects and they displayed similar pattern, further solidifying the bursty nature of the activity streams.

Kleinberg Burst Detection *(Klein)* - We use Kleinberg [24]'s burst detection method as our second segmentation model. It uses an infinite-state automaton to model the structure of bursty document streams over time. In order to adapt this formulation for our data, we calculated the *frequency* of occurrence of any activity, defined as the inverse of the *inter-arrival gap* as measured in days. These frequencies are modeled using the automaton as described in the paper, where each active timestamp (day in our case) is assigned an automaton state. In order to extract the bursts of activity, we take the highest automaton state assigned to a day. The days with the same states are grouped together to form bursts with same rates of emission. The output, therefore, is a series of bursts with varying intensities wherein the bursts contain either high- or low-frequency states.

Hidden Markov Model *(HMM)* - As our third model, we used a Multivariate Gaussian HMM which is a variant of the vanilla HMM except that the observation symbol probability distribution is assumed to be a Multivariate Gaussian process. The activity traces collected for the projects become the observation symbols that help in estimating the hidden states of the HMM. We experimented with different number of states and the best model was selected based on the likelihood of fit. In order to interpret the HMM states, we look at the average values of the model parameters (different activity counts). States with activity values less than a threshold were labelled as *"dormant"*, and the rest were labelled as *"active"*. The trained model was used to predict the state sequences for a given activity sequence for each project timeline. Each day that was assigned any one of the 'active' states was noted for the projects. A contiguous run of these 'active days' (with a maximum gap of 3 days between each subsequent active days), was grouped together to form a burst. The creation of bursts in this way segments the project activity timeline into a set of bursts of activity that can be compared to the burst segmentations obtained by the methods described previously in this section.

4.2 Burst Model Comparison and Selection

In order to select the segmentation model that performs the best, we need to have a platform and an evaluation metric for measuring the quality of the bursts yielded by them. We attempt to compare the bursts obtained by the models above with a lexically coherent segmentation of the project timeline using Beeferman et al. [2]'s P_k as the comparison metric.

Lexical Cohesion - TextTiling Segments *(TT Segments)* - The models described in the previous section considered only the raw activity counts to estimate a unit of work. Another approach is to examine the conversation among project developers, and consider a stretch of consecutive days with similar conversation topics as a *"lexically coherent burst"*. Activity bursts that are very similar to these lexical segments would not only represent maximum activity but also represent activity towards similar goals and hence can be inferred to be a coherent unit of work.

The lexical cohesion model that we use is a well-established method of text segmentation known as **TextTiling** [22]. It was designed to segment texts into coherent units reflecting the subtopical structure. The underlying assumption in this model is that a shift in the term distribution in text indicates a shift in the underlying topic of the text. The method starts by passing a sliding window over a representation (vector-space in our case) of the text. At each position of this window, a similarity score (cosine similarity) is computed between the upper and the lower region of the window. For example, at sliding window position i, the units from $i - k$ through i are compared to the units from $i + 1$ to $i + k + 1$, where k is one of the parameters of the algorithm. Scores for all possible positions i are calculated and recorded. For each i, a new score called the *depth score* is calculated that gives an insight into how sharp a change occurs on both sides of the window at position i. This is done by looking at the similarity scores to the left of i as long as it is increasing. This peak value is subtracted from the score at i to get the left height. The same process is repeated to get the right peak value and the right height. The two heights are added to give the depth score at i. A segment boundary is predicted at i if its depth score exceeds a threshold.

For our data, we collected the following textual information: issue comment text, issue titles, pull request titles, pull request comments, commit messages and commit comments. These texts were combined for each day, cleaned, stemmed and converted to word-vectors. Therefore each day of the project timeline was represented as a word vector. We set the value of k as 1, thereby comparing each adjacent day vectors for similarity. Segment boundaries were drawn on days when the depth score on that day exceeded a threshold. The threshold we used was $\mu - \frac{\sigma}{2}$, where μ and σ are the mean and the standard deviation of the depth scores for that project.

Beeferman's $\boldsymbol{P_k}$ - In order to evaluate the alignment between different segmentation methods, we use the evaluation metric $\boldsymbol{P_k}$ by Beeferman et al. [2]. It is commonly used to compare the segmentation boundaries predicted by a pair of models and find how similar or dissimilar they are. It is defined as the probability that "a pair of [events within a stream] at a distance of k [events] apart are inconsistently classified; that is, for one of the [models] the pair lies in the same segment, while for the other it spans a segment boundary." A Lower P_k value indicates a higher overlap and is therefore preferred. The ideal value of k, as suggested in their paper, is half the average segment length. Table 1 shows the pairwise P_k values for each pair of the non-lexical segmentation method to uncover the similarity of the segments they predict. Since the value of k depends on which segmentation method is selected as the first/reference method (it is the average reference segment length), we compute the P_k in each direction for a pair and take the average value. As we can see, HMM-Klein and Max-Sum-Klein have a high P_k indicating that the segmentation boundaries drawn by them are distinct. However, the low value for HMM and Max-Sum suggest that they predict similar segment boundaries. This is primarily because in both cases, the states estimated are segmented into binary 'active' and 'dormant' classes, where clusters of 'active' class days form a burst.

Model Selection - In order to find the most coherent segmentation, we first identified each segmentation's relative cohesion with the text stream. Table 2 shows the average pairwise P_k values of each method with the TT segments. It also shows the average number of text segments present in one segment of the non-lexical burst model, indicating the extent to which bursts represent one coherent focus. As we can see, Klein has the worst P_k value and an average of 7 text segments per Klein burst. Between HMM and Max-Sum, HMM has a slightly lower P_k but is substantially better in terms of burst coherence, containing approximately 1 segment per HMM burst whereas Max-Sum has almost 2 segments per burst. All three methods have different average segment lengths. We computed the P_k values after controlling for the segment length and the results were similar. Therefore, HMM predicts work-units that are more lexically coherent than the units predicted by Max-Sum. In comparison to the Max-Sum model which is a parameter free model, HMM is more interpretable. 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. It, therefore gives a better view into the characteristics of the data-stream. These reasons motivate us to choose HMM as an optimal segmentation method for our dataset. We use the burst length (duration in days) as a measure of *'productivity'* that serves as our success measure impacted by coordination in open source. This is with the intuition that a productive burst would involve people working efficiently and hence completing the tasks in a short span of time consistent with previous congruence studies [7, 10].

4.3 Coordination - Socio-Technical Congruence

In order to address our second research question, we attempt to operationalize the measures of socio-technical congruence and the related control variables [9] to investigate their effect on productivity (burst duration length). In a software project, task dependencies drive the need to coor-

Model 1	**Model 2**	**Average P_k**
HMM	Klein	0.52
HMM	Max-Sum	0.20
Max-Sum	Klein	0.53

Table 1: Non-lexical pairwise P_k values. Lower values indicate higher overlap.

Model 1	**Average # TT Segments Per Burst**	**Average P_k with TT Segmentation**
HMM	1.3	0.35
Max-Sum	1.8	0.38
Klein	7.2	0.41

Table 2: P_k values between event segmentation approaches and TT segmentation. Lower P_k values indicate higher alignment with text segmentation and number of text segments per burst indicate higher lexical cohesion.

dinate work activities. One of the most important questions about such coordination is who must coordinate with whom in order to get the work done. Congruence 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 us a measure of how much of the coordination that was required to happen, did happen. Intuitively, the hypothesis is that when a matter arises in which all the people with relevant expertise are actively participating, they can dispense with the matter more quickly. We take inspiration from the experiments done by Cataldo et al. [9] for their commercial dataset and adapt their measures in a way that makes them suitable for open-source projects.

4.3.1 Congruence the Traditional Way:

Cataldo et al. [8] introduced the idea of congruence in software projects, identifying three main components to estimating this measure: 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*). It is calculated as the ratio of the actual coordination to the required coordination.

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

Task Assignment Matrix (T_A): This is formulated as a people by task matrix where a one in the cell ij indicates that $person_i$ is assigned to $task_j$.

Task Dependency Matrix (T_D): This is the adjacency matrix (task by task matrix) of the dependency graph of the tasks where a non-zero entry in the cell ij indicates that $task_i$ is dependent on $task_j$.

C_R is calculated as shown in equation 1 where a non-zero entry in cell ij indicates that $person_i$

should coordinate with $person_j$ while working on the assigned tasks.

$$C_R = T_A * T_D * T_A' \tag{1}$$

Actual Coordination ($\boldsymbol{C_A}$): This is formulated as a people by people matrix. A non-zero entry in the cell ij indicates that $person_i$ actually coordinated, by different means of coordination available, 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 2.

$$Congruence(C_R, C_A) = \frac{\sum(C_A \wedge C_R)}{\sum(C_R)} \tag{2}$$

4.3.2 Congruence the GitHub Way:

Cataldo et al. [9] calculated four measures of congruence, each for a different means of communication among people. We attempt to operationalize two of them, *MR Congruence* and *Structural Congruence* for each burst in our final set of GitHub projects; the other two did not map well to GitHub data. The basic definition of the matrices, explained in the previous section, remains the same. The way each of them are translated to be valid for GitHub projects is described below. These were calculated for each active burst for all the projects, where the files committed in the burst are treated as the "tasks".

$\boldsymbol{T_A[i][j]}$ $\rightarrow$ A non-zero value indicates that $file_j$ was committed by $person_i$ in the burst.

$\boldsymbol{T_D[i][j]}$ $\rightarrow$ We use the FCT (Files Changed Together) method, utilized by Cataldo et al. [9], to build an undirected dependency graph for the GitHub projects. It is constructed by looking at the commit history of the files changed in the burst, with the assumption that the files that are committed together are logically related to each other. Each node in this graph represents a file. A non-zero value in $T_D[i][j]$ indicates that $file_i$ and $file_j$ have a connecting edge in the co-commit graph and are dependent on each other.

$\boldsymbol{C_A[i][j]}$ **for MR Congruence** $\rightarrow$ 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.

$\boldsymbol{C_A[i][j]}$ **for Structural Congruence** $\rightarrow$ In their implementation of structural congruence, Cataldo et al. [9] find the people belonging to the same organizational team and mark them as candidates who communicate with each other via periodic team meetings or informal discussions. Since we do not have well defined formal teams on GitHub we instead infer virtual teams as people who have worked in related parts of the code. We partition the co-commit graph using the fast unfolding method by Blondel et al. [3], in order to find communities or groups of the files that change together and are highly interdependent. People committing the files that belong to the same group are very likely to have communicated with each other or commented and reviewed the commits made by others in the group. Therefore, a non-zero value for $C_A[i][j]$ represents that

$person_i$ and $person_j$ have committed files that belong to the same group (network communities) at some point and belong to the same virtual team.

4.4 Control Measures

Cataldo et al. [8] identify numerous factors that impact the productivity (measured as the resolution time of Modification Requests (MRs)) of the commercial software systems. In later work, they use these factors as control measures in regression models [9]. Since these factors are designed with commercial systems in mind, we cannot directly translate all of them to the open-source systems. We excluded *Temporal Dependency, Priority, Re-assignment, Release* and *Multiple Location* as they were difficult to apply to GitHub data. For example, GitHub projects do not mark *priority* consistently, if at all; there is no accurate way of knowing the *geographical locations* of the contributors and the release practices are quite different across projects. There were similar blocks for other excluded factors. The following list describes the implementation of the subset of factors that we were able to operationalize for GitHub.

1. **Change Size** - This measure approximates the actual amount of work done and is calculated as the total number of files that were committed in the burst.

2. **Team Load** - This is the measure of average work load and is calculated as the number of active issues per committer in the burst.

3. **Component Experience** - This measure is calculated as the average number of times the active committers of the burst have 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 3 where c_{ij} represents the number of commits made by $person_j$ to the $file_i$ before the current burst started.

$$component\_experience = \frac{\sum_{j=p_1}^{p_k} \sum_{i=f_1}^{f_n} (c_{ij})}{(k * n)} \tag{3}$$

4. **Tenure** - For the active committers in the burst, this measure represents the average number of days that they have been a part of the GitHub project at the time of completion of that burst.

5. **Activity per Person** - This measure is calculated as the average number of *commits* and *comments* made per active contributor in the burst.

6. **Number of Teams** - As explained in Section 4.3.2, we identified distinct communities or group of files from the co-commit network. This measure is a count of the distinct number of these network communities involved in the burst. It signifies the amount of the cross-team collaboration effort required for the files being committed in the burst.

4.5 Impact on Productivity

We use a linear regression model in order to understand the impact of congruence and the control measures on the prediction variable - *Burst Duration Length* (in days). We estimated two models: one without the congruence measure (Model I) and one with the congruence measure (Model II). We dropped bursts in which only one developer committed, since congruence is not a valid measure in those cases. Our final dataset consisted of 60,458 bursts across 9,960 projects. The measures *"Team Load"* and *"Activity per Person"* were highly correlated and thus it made sense to include only one of them. In order to be consistent with the experiments in Cataldo et al. [9], we settled on including *Team Load*. *Structural Congruence* was not a significant indicator and was highly correlated to *MR Congruence*. Therefore, we exclude that measure too from our final model. An analysis of the pair-wise correlations of the final list of control measures indicate no significant collinearity among them.

	Model I	**Model II**
(Intercept)	-1.21^{**}	-0.08^{+}
Team Load	0.95^{**}	0.97^{**}
Change Size	0.07^{**}	0.07^{**}
Component Experience	-0.02^{**}	-0.02^{**}
Number of Teams	3.1^{**}	3.2^{**}
Tenure	-0.0006^{**}	-0.0006^{**}
MR Congruence		$\mathbf{-2.5^{**}}$
N	60458	60458
Adjusted R^2	0.291	0.296

($^{**}p < 0.0001$, $^{+}p < 0.5$)

Table 3: Effects on Burst Duration Length

5 Results

Table 3 shows the results of our regression experiments. Model I is a baseline regression model that considers only the control measures. Consistent with the experiments in Cataldo et al. [9] and the previous empirical work in software engineering [17], factors such as familiarity with the software components (*component experience*) and general programming experience (*tenure*), help in faster completion of the work items and lead to shorter burst duration, as suggested by their negative parameter estimates. However, the coefficient for *tenure* is very small indicating its low effect on the prediction. Herbsleb and Mockus [23] found that as the number of people involved or the size of a change required increases, the resolution time for modification requests (MRs) also increases. As in Cataldo, *Number of Teams* and *Change Size* are significant with a positive coefficient in our experiments. The similar results lend weight to the idea that the issue resolution time in Cataldo's commercial setting is comparable to longer burst length in GitHub projects. On the other hand, unlike Cataldo et al. [9] where *Team Load* was not a significant factor, we found it to be significant and positive; i.e. as the number of issues per person increases, the burst duration also increases, implying that it takes longer to complete the active tasks of the burst.

Model II introduces the measure of *MR Congruence* in our analysis. It has a statistically significant effect on the burst duration length. The estimated coefficient is negative, indicating that if congruence is high, i.e. the coordination requirements for the active tasks in the burst are met, then the tasks are completed more quickly, and thus the burst is shorter. This is in line with the previous findings on the effect of congruence and the resolution times of the MRs of large commercial systems [9, 7, 8]. Therefore, it validates our expectation that congruence has a role to play in explaining the effect of coordination on the productivity of open-source GitHub projects. If the required people coordinate, the burst duration might be shorter. However, its role in accounting for the burst duration is small and there is only a slight increase in the R^2 value or the model fit (roughly $29\% \rightarrow 30\%$). Other control measures retain their significance and the coefficient strength, as in Model I, and therefore have a similar effect on the burst length.

The code and data are available in a separate data release publication.[**?**]

6 Discussion

Even though congruence only accounts for about one percent of the model fit, it does not mean that it is not important. The low R^2 value suggests that perhaps there is not a huge variability in congruence values. We found that the larger the number of people, the lower the congruence is on average, but most of the time, a small number of people are active in a burst. Therefore, there are certain times when it is important in explaining the productivity of the participants, indicating it to be salient feature over all.

Measuring productivity in open-source projects is difficult because of the lack of a clear measure of success, as well as the diversity, informality and the voluntary nature of collaboration. Open source developers sometimes use ad-hoc measures, like number of open bugs or volume of activity, to assess how "serious" a project is [13]. Our measure of productivity has the potential to quickly and comprehensively assess whether a project is working efficiently at any given time. Further, our end-to-end pipeline of identifying the bursts of activity, calculating the control measures and operationalizing congruence can be leveraged to make aspects of collaboration visible in ways that could help address the productivity problems.

The HMM helped us separate out interesting episodes in a project's timeline for studying collaboration, but we believe it has other potential uses for identifying relevant events and phases as a project evolves. For example, there could be distinct phases of recruitment and retention of developers or there might exist diverse patterns of work on different artifacts (code, documentation, tests) over the project lifecycle. The segmentation of the project timeline into bursts can also be leveraged to visualize and study different collaborative behaviors by focusing on intervals of time that are most important. Since GitHub provides no direct way of associating commits (i.e. code changes) with the reported issues (i.e. bug reports and feature requests), these bursts can serve as a good approximation of this association to facilitate other coordination studies in open-source.

There have been studies in the past showing that individuals have difficulties identifying non-explicit task interdependencies and dynamic coordination requirements over time [38, 8]. Many collaboration tools have been designed to identify and exhibit these dependencies to the developers. However, their adoption has mostly been limited to either commercial or a single large open-source software system, where the identification of such dependencies is tractable. Their employment to large-scale open-source platforms like GitHub has been minimal at the least. Some prior tools in this vein include dependency checkers such as Tukan [37], CollabVS [15], and Palantìr [35]. They perform basic code analysis to identify dependencies among artifacts and warn developers when they are changed by others. There are also tools that leverage not only the technical but also the social information of the software projects. Sarma et al. [36] developed *'Tesseract'*, an interactive visualization tool that leverages the socio-technical information computed by congruence to show the relationships among different project entities that could help the developers align their interactions at work. Along similar lines, de Souza et al. [14] developed an Eclipse plugin *'Ariadne'* that analyzes software projects for technical dependencies among components and translates them into social dependencies among developers. The technique suggested in this paper could provide a good guideline in adapting and extending these tools to GitHub.

We demonstrated this technique for a varied set of PyPi projects, but it is generic enough to be applied to other domains on GitHub. We envision, for example, a GitHub plugin that would enable the developers of the projects to analyze the project's current productivity or the congruence values in the recent episodes of activity to detect coordination breakdowns and take appropriate steps. The information from the co-commit graphs and the user-file associations can be leveraged to identify the socio-technical dependencies for the projects. The *'Number of Teams'* control measure could also be useful as a network partitioning method to identify clusters of interdependent files and the people associated with it. This clustering of people into teams might give the developers an insight into the implicit organization of the project and identify the team that they belong to. This would especially be helpful for small and medium sized projects that have no commercial backing or official teams.

All in all, platforms like GitHub could make use of a productivity measure and its components, through the techniques presented here, in order to bring more situational awareness about productive collaboration into the user experience on these platforms.

Future work - Socio-Technical Congruence has been a very useful metric and captures *who* should coordinate for the successful completion of different tasks quite well. However, it does not capture the matters of *how* they should collaborate. This is apparent by the low fit of our regression model. In future, we would like to understand *how* the coordination is carried out. Open-source developers coordinate and communicate with each other on myriad platforms of communication ranging from bug reports, issue threads, IRC, mailing lists and so on. We would like to build a pipeline to that would help us understand collaboration beyond what congruence can explain on open-source projects by looking at the developer conversations and identifying indicators for user behaviors that serve as good predictors of project health and success.

7 Conclusion

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 a successful completion of the tasks. We compared and used different burst detection models 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 experiment showed that socio-technical congruence and related control variables had a significant, albeit subtle, impact on the productivity of the active phases of the projects.

References

[1] Albert-László Barabási. *Bursts: the hidden patterns behind everything we do, from your e-mail to bloody crusades*. Penguin, 2010.

[2] Doug Beeferman, Adam Berger, and John Lafferty. Statistical models for text segmentation. *Machine learning*, 34(1):177–210, 1999.

[3] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10): P10008, 2008.

[4] Francesco Bolici, James Howison, and Kevin Crowston. 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.

[5] Francesco Bolici, James Howison, and Kevin Crowston. Stigmergic coordination in FLOSS development teams: Integrating explicit and implicit mechanisms. *Cognitive Systems Research*, 38:14–22, 2016. ISSN 13890417. doi: 10.1016/j.cogsys.2015.12.003. URL `http://dx.doi.org/10.1016/j.cogsys.2015.12.003`.

[6] Tyson R Browning and Ranga V Ramasesh. A survey of activity network-based process models for managing product development projects. *Production and operations management*, 16(2):217–240, 2007.

[7] Marcelo Cataldo and James D Herbsleb. Coordination breakdowns and their impact on development productivity and software failures. *IEEE Transactions on Software Engineering*, 39(3):343–360, 2013.

[8] Marcelo Cataldo, Patrick A Wagstrom, James D Herbsleb, and Kathleen 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*, pages 353–362. ACM, 2006.

[9] Marcelo Cataldo, James D Herbsleb, and Kathleen 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*, pages 2–11. ACM, 2008.

[10] Marcelo Cataldo, Audris Mockus, Jeffrey A Roberts, and James D Herbsleb. Software dependencies, work dependencies, and their impact on failures. *IEEE Transactions on Software Engineering*, 35(6): 864–878, 2009.

[11] Samridhi Sree Choudhary, Christopher Bogart, Carolyn Pennstein Rosé, and James D. Herbsleb. Modeling productivity in open source github projects: A dataset and codebase. https://doi.org/10.1184/R1/6397013.

[12] Bill Curtis, Herb Krasner, and Neil Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.

[13] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim 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*, pages 1277–1286. ACM, 2012.

[14] Cleidson R de Souza, Stephen Quirk, Erik Trainer, and David F Redmiles. Supporting collaborative software development through the visualization of socio-technical dependencies. In *Proceedings of the 2007 international ACM conference on Supporting group work*, pages 147–156. ACM, 2007.

[15] Prasun Dewan and Rajesh Hegde. Semi-synchronous conflict detection and resolution in asynchronous software development. In *ECSCW 2007*, pages 159–178. Springer, 2007.

[16] Anwar I Elwalid and Debasis Mitra. Effective bandwidth of general markovian traffic sources and admission control of high speed networks. *IEEE/ACM Transactions on Networking (TON)*, 1(3):329–343, 1993.

[17] J. Alberto Espinosa. *Shared Mental Models and Coordination in Large-scale, Distributed Software Development*. PhD thesis, Pittsburgh, PA, USA, 2002. AAI3065743.

[18] Victor S Frost and Benjamin Melamed. Traffic modeling for telecommunications networks. *IEEE Communications Magazine*, 32(3):70–81, 1994.

[19] Gabriel Pui Cheong Fung, Jeffrey Xu Yu, Philip S Yu, and Hongjun Lu. Parameter free bursty events detection in text streams. In *Proceedings of the 31st international conference on Very large data bases*, pages 181–192. VLDB Endowment, 2005.

[20] Raghu Garud, Arun Kumaraswamy, and Richard Langlois. *Managing in the modular age: architectures, networks, and organizations*. John Wiley & Sons, 2009.

[21] Amir Hossein Ghapanchi, Aybuke Aurum, and Graham Low. A taxonomy for measuring the success of open source software projects. *First Monday*, 16(8), 2011.

[22] Marti A Hearst. Multi-paragraph segmentation of expository text. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 9–16. Association for Computational Linguistics, 1994.

[23] James D. Herbsleb and Audris Mockus. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on software engineering*, 29(6):481–494, 2003.

[24] Jon Kleinberg. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery*, 7(4):373–397, 2003.

[25] Andrew J Ko and Parmit K Chilana. Design, discussion, and dissent in open bug reports. In *Proceedings of the 2011 iConference*, pages 106–113. ACM, 2011.

[26] Anders Krogh, Michael Brown, I Saira Mian, Kimmen Sjölander, and David Haussler. Hidden markov models in computational biology: Applications to protein modeling. *Journal of molecular biology*, 235 (5):1501–1531, 1994.

[27] Theodoros Lappas, Benjamin Arai, Manolis Platakis, Dimitrios Kotsakos, and Dimitrios Gunopulos. On burstiness-aware search for document sequences. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 477–486. ACM, 2009.

[28] Alan MacCormack and Roberto Verganti. Managing the sources of uncertainty: Matching process and context in software development. *Journal of Product Innovation Management*, 20(3):217–232, 2003.

[29] Jennifer Marlow, Laura Dabbish, and Jim Herbsleb. Impression formation in online peer production: activity traces and personal profiles in github. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 117–128. ACM, 2013.

[30] Jamie F Olson, James Howison, and Kathleen M Carley. Paying attention to each other in visible work communities: Modeling bursty systems of multiple activity streams. *International Conference on Social Computing (SocialCom)*, pages 276–281, 2010. doi: 10.1109/SocialCom.2010.46.

[31] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[32] Christoph Riedl and Anita Williams Woolley. Teams vs. crowds: A field test of the relative contribution of incentives, member ability, and emergent collaboration to crowd-based problem solving performance. *Academy of Management Discoveries*, 3(4):382–403, 2017.

[33] Bruno Rossi, Barbara Russo, and Giancarlo Succi. Analysis of Open Source Software Development Iterations by Means of Burst Detection Techniques. *Proc. International Conference on Open Source Systems (OSS)*, 299:83–93, 2009. doi: 10.1007/978-3-642-02032-2.

[34] Walter L Ruzzo and Martin Tompa. A linear time algorithm for finding all maximal scoring subsequences. In *ISMB*, volume 99, pages 234–241, 1999.

[35] Anita Sarma, Zahra Noroozi, and André Van Der Hoek. Palantír: raising awareness among configuration management workspaces. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 444–454. IEEE, 2003.

[36] Anita Sarma, Larry Maccherone, Patrick Wagstrom, and James Herbsleb. Tesseract: Interactive visual exploration of socio-technical relationships in software development. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 23–33. IEEE, 2009.

[37] Till Schümmer and Jörg M Haake. Supporting distributed software development by modes of collaboration. In *ECSCW 2001*, pages 79–98. Springer, 2001.

[38] Manuel E Sosa, Steven D Eppinger, and Craig M Rowles. The misalignment of product architecture and organizational structure in complex product development. *Management science*, 50(12):1674–1689, 2004.

[39] Margaret-Anne Storey, Leif Singer, Brendan Cleary, Fernando Figueira Filho, and Alexey Zagalsky. The (R) Evolution of Social Media in Software Engineering. *FOSE 2014 Proceedings of the on Future of Software Engineering*, pages 100–116, 2014. doi: 10.1145/2593882.2593887. URL `http://dl.acm.org/citation.cfm?doid=2593882.2593887`.

[40] Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th international conference on Software engineering*, pages 356–366. ACM, 2014.

[41] Jason Tsay, Laura Dabbish, and James Herbsleb. 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*, pages 144–154. ACM, 2014.

[42] Michail Vlachos, Christopher Meek, Zografoula Vagena, and Dimitrios Gunopulos. Identifying similarities, periodicities and bursts for online search queries. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 131–142. ACM, 2004.

[43] Yunyue Zhu and Dennis Shasha. Efficient elastic burst detection in data streams. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 336–345. ACM, 2003.