Noname manuscript No.
(will be inserted by the editor)

Tracing Distributed Collaborative Development in Apache Software Foundation Projects

Mohammad Gharehyazie · Vladimir Filkov

July 21, 2016

Abstract Developing and maintaining large software systems typically requires that developers collaborate on many tasks. During such collaborations, when multiple people work on the same chunk of code at the same time, they communicate with each other and employ safeguards in various ways. Recent studies have considered group co-development in OSS projects and found that it is an essential part of many projects. However, those studies were limited to groups of size two, *i.e.,* pairs of developers.

Here we go further and characterize co-development in larger groups. We develop an effective methodology for capturing distributed collaboration beyond groups of size two, based on synchronized commit activities among multiple developers, and apply it to data from 26 OSS projects from the Apache Software Foundation. We find that distributed collaborations is prevalent, but not as frequent as expected. We also find that while in distributed collaborative groups, developers' behavior is different than when programming alone, *e.g.,* high developer focus on specific code packages associates with lower team participation, while packages with higher ownership get less attention from groups than from individuals. Finally, we show that productivity effort during co-development is more often lower for developers while they co-develop in groups. To verify our results we use both quantitative and qualitative methods, including a developer survey.

We conclude that these methods and results can be used to understand the effects of the collaborative dynamic in OSS teams on the software engineering process. Our code, along with our datasets and survey is available at `http://www.gharehyazie.com/supplementary/teamwork/`

M. Gharehyazie, and V. Filkov
Computer Science Department
University of California, Davis
Davis, CA, 95616
E-mail: gharehyazie@ucdavis.edu, filkov@cs.ucdavis.edu

Keywords Teamwork, Distributed Collaborative Development, OSS

1 Introduction

Teamwork's advantages are enjoyed in many walks of life, by humans [14] and other animals [66]. And those advantages become even more apparent in the digital realm, where people don't even have to be geographically near. Teams can be pre-meditated and serve a concrete purpose, *e.g.,* two people carrying a couch up the stairs, or group of undergrads building Facebook; then the outcome clearly benefits from the team's existence. Not surprisingly, team forming is taxing; those who team up incur costs related to the overhead of finding partners and coordinating with them. The larger the teams the higher this overhead [18,58]. In fact, managing large teams effectively is a top reason for the invention of organizational structure, and the subject of much managerial science research [13, 50].

But in many occasions teams also grow organically, without much explicit organization at any given growth step, thus reducing costs related to teaming up [36]. Examples include ants and bees swarming, schools of fish changing direction on a dime [19], and Open Source Software teams. In many ways, developing Open Source Software (OSS) code depends on teamwork, as many developers work toward the same goal [15]. Often teams in OSS form organically, as needed, though persistent organizational structures exist in some of them. [1] Teaming up in OSS serves many purposes, including finishing tasks which require many person-hours, and tasks which require varied expertise. It is also often implicit. Large OSS are known to exhibit latent communities which follow a hierarchical structure [7], organized around the socio-technical activities of developers. But how prevalent are those implicit, self-organized teams? Are they persistent? Do some people have higher preference for working collaboratively because of their work styles? And does working on a team impart any beneficial effects on the team members in terms of efficacy?

Leveraging the availability of trace data from publicly available OSS projects, we have recently studied in 6 OSS projects the benefits of OSS developer collaborations by tracing pairwise synchronous code development, i.e., when two developers work on the same file around the same time [70]. In that setting, we found that two-person teams exist and are strongly affecting developer productivity. There, we left open the cases of synchronous code development in sets of sizes 3 and more people, due to lacking the technology to discover larger teams. The difficulty in scaling up the study from teams of size 2 to larger

[1] Linus Torvalds runs the Linux project in a more centralized fashion, depending on his lieutenants for decisions regarding which new code filters up to him.

teams lies in the fact that it is non-trivial to implicate people in teamwork when dealing with OSS projects. This is true for many obvious reasons such as lack of central management and users having free will to work with or on whatever strikes their fancy at any given moment. However, techniques arising from self-organizational studies and research in task synchronization between actors help in setting the frameworks for analyzing heterogeneous trace data abundant in empirical software engineering.

Here, we are interested in how collaborative distributed development scales beyond two people, to teams of three and more developers, when we operationalize it as synchronous distributed development. Our goal is to infer, or trace, putative teams from the large-scale data traces of developers' technical activities in Apache Software Foundation (ASF) OSS projects. We chose ASF for several reasons including ASF's multiple active and popular projects, and potential synergy with an already existing body of research [22, 67–72]. We build on prior work on latent communities [7] and code development collaboration [70]. From the latter we take a definition of distributed collaboration and extend it to larger groups, in a way which easily lends itself to use with trace data of developer activities.

Like in our precursor work,we narrow down the definition of collaboration so that we can practically detect them from trace longitudinal data of commits. To consider a group of developers as having a collaboration, their technical actions, *i.e.,* code contributions, should be able to affect those of others. Physical and temporal proximity of code snippets being worked on by different developers increase the chance of merge conflicts thereby making it more likely for one developer to affect the other's contribution. We leverage this intuition to identify putative distributed collaborative groups in this study.

> We call a putative *Collaborative Group (CoG)* a set of developers who can be identified via their code commits as working in close *code proximity* and *temporal proximity* to each other.

We chose our temporal and code proximity to be *one week* and a *package*, respectively. We will elaborate on these choices later in Section 4.3 extensively. With these parameters, our practical definition of a CoG becomes: "A set of developers working on the same set of packages within a time interval of 1 week from each other." From this point on, we interchangeably use collaboration, collaborative development, and distributed collaborative development, to refer to the above definition.

In the rest of the paper we describe how we used the above definition on longitudinal trace data from 26 ASF OSS projects to do the following.

- We develop a robust method for tracing putative CoGs from trace OSS data and use it to characterize distributed collaboration levels in 26 ASF projects;

- We show that developers in CoGs communicate and coordinate through the developer mailing lists;
- We find that developers spend a significant amount of time collaborating with other developers and that their contribution patterns differ during these collaborations;
- We measure developers' group collaboration levels, with respect to the amount of time, number of commits, and lines of code contributed, and find that not all people spend a significant amount of time collaborating with others, but in most cases, when they do collaborate, they commit more per unit time;
- We find that developers with higher focus on specific packages are also less likely to collaborate, while packages with higher ownership, as expected, are less likely the subject of distributed collaboration;
- We find that in terms of LOC added and deleted per commit, individual effort decreases during collaboration for 17 projects, while code growth decreases for 10 and increases for 2 projects, suggesting that while collaborating, developers arguably contend with problems of some increased complexity; and
- We ran a survey of ASF developers which provides input to our modeling assumptions, and anecdotal evidence for our results.

We describe the theory and our research questions in Section 2, followed by related works in Section 3. The methodology, results, threats, and conclusions, are in Sections 4, 5, 6, and 7, respectively.

2 Theory and Research Questions

Our first goal is to find putative CoGs and quantify their prevalence in our sample of ASF OSS projects. We do that by developing an algorithm for reasonably capturing the above definition of putative CoGs. In addition to their counts, we also want to know how their prevalence stacks against a random model of contributions. Since OSS projects also make publicly available some communication records between developers, validation of theses putative CoGs is plausible.

Research Question 1: How prevalent are putative CoGs, of size 3 and more? Do they occur more frequently than by chance? And is there evidence in the trace data that they are not just quantitative artifacts but real teams?

Having a way to count CoGs, we next turn to understanding programmer's participation in those groups, and if their preference for participation changes over time. In other words, we want to understand how developers' split their contributions between periods of collaboration and solitary work. This would show how prevalent co-development is from a personal perspective.

Research Question 2: How frequently do developers work in groups? Does preference for working in groups vs working solo change over time?

Next we turn to questions having to do with the effects of group work on the developer performance and technical output. The amount of one's focus on a particular file vs. distributing their activities equally across multiple files is an important contribution measure for developers. It can be quantified using a developer's contribution entropy. Code ownership which measures the entropy of developer contributions to each file can be seen as a dual of developer focus. It has been shown that focused developers are less likely to introduce defects [55], and Windows binaries with higher ownership may include fewer defects [6]. Is it, then, reasonable to think that while in a group, a developer's focus may decrease and thus result in higher defects and lower productivity? So, in the next part we are interested to see how co-development correlates with developer focus and file ownership.

Research Question 3: Does distributed collaborative development correlate with developer focus? I.e., do more focused developers participate in fewer CoGs?How about package ownership, do packages that are subject of co-development display a particularly high or low code ownership?

And finally we wish to see how distributed collaboration associates with individual productivity. Two straightforward metrics of productivity [70] are the code effort and code growth per developer for each file, defined as 1. code effort: LOC added plus LOC deleted per commit. 2. code growth: LOC added minus LOC deleted per commit. The code effort per file describes the amount of work performed on each file at each commit, while the code growth per file explains how much the size of a file grows per commit. These metrics can tell us whether developers are working more or less per file during distributed collaborative development.

Research Question 4: Do developers experience a notable change in effort or code growth during co-development? In particular, is there a pattern to the change as suggested by the data, *i.e.,* in the same direction, over all projects and developers?

3 Related work

Most directly related to this paper is a recent study by Xuan and Filkov, in which synchronous co-development was studied among pairs of developers in 6 OSS projects using trace data from commits and mailing list posts [70]. The

co-development discovered was very strong, indicating that when collaborating together on the same set of files, developers communicate more frequently and *add* more lines of code than they *delete*. In this paper, we focus on all groups, and not just those of size 2. We aim to build upon this work by extending to groups of larger sizes and moving beyond files to packages as a notion of code proximity, which will allow us to identify collaborations that were previously undetected. This gives us a higher level view, where we will see groups which potentially form towards completion of specific tasks, rather than a set of pairwise collaborations that are not connected to each other. We also expand our study to a wider range of projects, allowing us to provide a more complete insight on collaborative development.

Pinzger and Hall utilize a similar concept of code proximity to develop a collaboration and communication visualization tool for Eclipse [53]. Their tool helps identify collaborators on certain components, however, their approach is less concerned with a concrete definition of collaboration, and rather leaves its definition at the hands of the user through the tuning of the tool's parameters. Jermakovics *et al.* also use the notion of code proximity to identify the collaboration network in software projects [33]. Their approach however disregards any notion of temporal proximity which can lead to identifying unrealistic long-term collaborations. Caglayan *et al.* have also developed a community detection based approach to identifying teams [10] but their approach makes identified teams less suitable for the study of developers interactions. As mentioned, there are very few works on quantifying collaboration in OSS, most of which we have discussed here. Still there is a certain lack of solid quantitative methods to identify collaboration across OSS [65] which our work seeks to address.

With respect to social organization, Bird *et al.* have shown that developers organize themselves into groups and communities based on their social activities [7]. They have also shown that there is a socio-technical congruence between the software artifact modularity, and its developers' social groups. Panichella *et al.* extended this study by studying how these groups evolve and change over time [52]. Their study shows team structure is not stable and evolves over time, with respect to cohesiveness of files teams work on. Both of these studies are based on social networks extracted from the developer mailing lists; our current study is complementary, in that we look at groups arising from synchronous commit patterns. Robertsa *et al.* also used communication patterns and identified a strong group at the core of the Apache HTTP project [57]. Damian *et al.* studied communication and coordination patterns in an IBM team and found that task related social-network are ever-evolving and different from their initial conception [17]. Kakimoto *et al.* use Social Network Analysis to study the communication patterns in SourceForge projects and find an intensification of communications among members with different roles around the time of a release [34].

Hertel *et al.* studied the motivations of developers for joining specific subsystems [31] through a survey of 141 Linux Kernel developers. They observe that *valence*, *instrumentality* and *self-efficacy* are driving developers towards increased participation, patch submission and time spent in the project. However beneficial teaming up might be, Mockus showed organizational volatility increases the probability of software defects [44]. In terms of overhead incurred for teaming up, Adams *et al.* studied the Brooks' law effect on OSS [1]. They find that coordination costs increase only in a specific phase of a project and after that, it becomes "quasi-constant". Nagappan *et al.* define a set of metrics based on organizational structure and show that they are able to predict defect-proneness better than classic code metrics such as code churn [46]. While their metrics are certainly interesting, they are defined based on an established well defined organizational structure, and many of the metrics are not applicable to our implicit and non-hierarchical definition of CoGs.

Dabbish *et al.* interviewed a group of GitHub developers and found that transparency in large development efforts support efficiency and collaboration [16]. In a case-study, Herbsleb and Gringer [28] show that communication is essential to code development and that distribution of activities can hinder communication. Cataldo and Herbsleb [12] show that a gap between coordination requirements and actual coordination can increase software failures. Luther *et al.* identified frequent communication and high activity of members and leaders as a success factor in online collaborations, including OSS [39]. Nakakoji *et al.* describe OSS projects as participative system and hypothesize the that the change in participants' perceived values in a project may characterize the evolution of that project and its community [47].

A topic that is highly tangential to collaboration is *code ownership*. Rahman and Devanbu have studied the effect of code ownership in OSS [55] and find that defects are more likely from contributions of a single developer. Bird *et al.* studied proprietary software *viz.* Windows 7 and Vista [6]. Both studies above are in agreement and report that files or binaries with many minor contributors are more likely to contain defects. Focault *et al.* attempted to replicate the results by Bird *et al.* [6] in OSS, but their results was in contrast to the previous works' findings [20]. However, the difference in their methodology and corpora studied, makes the comparison of their results a little controversial. *Developer focus* is another issue that can be seen as the dual of code ownership. Posnett *et al.* show that these two measures can be unified under a more generalized view [54]. We use those measures in this paper.

While, to our knowledge, there is no direct study of the correlation between group collaboration and performance and productivity in self-organized teams, in the field of managerial science the concept of teamwork and its correlation with performance has been touched on by Brooks [9], Kuipers and de Witte [37], Moe *et al.* [45], and others. They find that improved cooperation is needed to gain higher productivity, but team size does not figure linearly.

They also show that increasing task delegation reduces defects in a team's product. These results are for centrally managed, well developed teams.

More broadly, our work can be contextualized into research on *collaborative Software Engineering*, where the emphasis is on the mechanisms and outcomes of collaboration [65]. Scacchi has produced an overview of empirical studies of OSS projects, focusing on collaboration practices at individual, project and cross-project level [61]. Prior work in this field has addressed various challenges, such as how to leverage others' experience, and what an effective collaboration entails [42], including the tools and platforms for collaborating in software development as enabling technologies [38]. Along those lines, Avritzer and Paulish provide a comparison of common processes utilized in multi-site software development [3].

Much work has been done on understanding and ameliorating the potential harmful effects of distance collaboration in the area of *Global Software Engineering* where the development teams are geographically distributed [30]. Holmstrom *et al.* use a case study to identify and highlight the key challenges in global software development, specially in regards to temporal, geographical and socio-cultural distances [32]. Sarma *et al.* outlined the main challenges organizations face in achieving *congruence* [60]. Grechanik *et al.* study the shortcomings of outsourced quality assurance teams and argue that new approaches are needed to overcome these issues [24]. Herbsleb *et al.* measure the costs of cross-site development vs same-site work and find out that cross-site work takes longer and requires more people to complete [29]. Al-Ani and Edwards studied the communication patterns of a fortune 500 company and found that the overhead for synchronous communication is unacceptably high [2]. They also show that the patterns of communication evolve from the initial inception to address the development needs. These findings in part motivated us to study the patterns of communication and collaboration among OSS developers.

Interestingly, Takhteyev and Hilts look into the distribution of users in the GitHub ecosystem and find a strong local bias in contribution and attention [63]. This suggests that the challenges of distributed development in many OSS is less about geographical distance and more about remote contributions and lack of physical presence. Nguyen *et al.* also showed that geographical distribution did not play a significant role in coordination response times [49], suggesting that coordination overheads are less about the distance and more about the communication itself. Nakakoji *et al.* describe two different types of communication in software development: *coordination communication* and *expertise communication* [48]. They provide nine outlines for properly supporting expertise communication in OSS projects. Redmiles *et al.* introduce *Continuous Coordination* as a paradigm for coordination and communication that addresses some of the issues faced in global software engineering [56]. Later, Sarma *et al.* evaluated the continuous coordination tools available and their usefulness through an evaluation framework called DESMET [59]. Lastly, Carmel [11] presents some of the successful practices and techniques to make

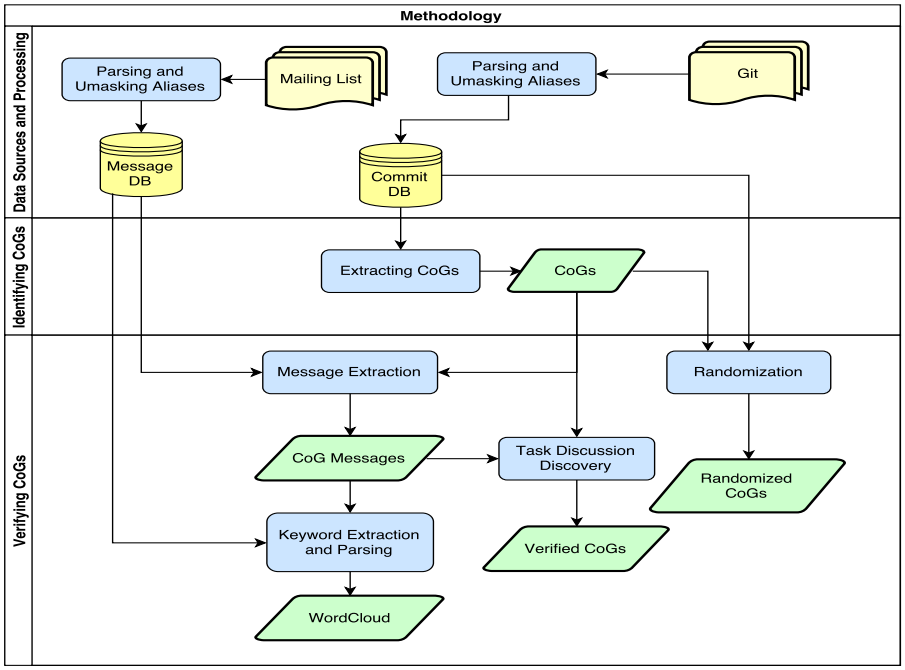

Fig. 1. The process diagram of the data gathering, identification and verification steps as described in the paper.

the best of distributed software development, while Herbsleb [27] outlines a desired vision of global development and expresses the need for a systematic understanding of the driving force of an effective coordination.

4 Methodology

Figure 1 provides an overview of the process of mining and parsing data, extracting CoGs and multiple verification steps, as described below.

4.1 Mining Source Code Repositories

We mined the git source code repositories for 26 ASF projects; summary is in Table 1. Data gathering on the mailing lists and the repositories was performed at different times and the times shown in Table 1 represents the intersection of both of these datasets' date range. All of the selected projects have at least 5 developers and 2 years worth of commit history that is maintained in their Git repository.

Table 1. List of OSS projects used in this study. The end date in most of the projects is the date that data collection was performed.

Projects	Devs	Start	End	Projects	Devs	Start	End
abdera	13	2006-06	2011-12	ivy	9	2005-06	2012-01
activemq	28	2005-12	2012-01	log4j	18	2000-11	2012-01
ant	44	2000-01	2012-02	log4net	7	2004-01	2011-12
avro	12	2009-04	2011-12	log4php	9	2004-01	2012-01
axis2_c	24	2005-09	2010-02	lucene	41	2001-09	2011-01
camel	31	2007-03	2012-01	mahout	15	2008-01	2012-01
cassandra	13	2009-03	2011-12	nutch	16	2005-01	2012-01
cayenne	20	2002-03	2012-01	ode	17	2006-05	2011-12
cxf	45	2005-07	2012-01	openejb	38	2002-01	2012-01
derby	35	2004-08	2012-01	pluto	24	2003-09	2011-09
hadoop_hdfs	25	2009-05	2011-06	solr	19	2006-01	2011-03
harmony	25	2005-09	2011-07	wicket	24	2004-09	2011-12
hive	18	2008-09	2012-01	xerces2_j	33	1999-11	2012-01

The `git log` command gives us a complete record of each commit, along with its id, date, and committer. We then parse the output of `git show` for each commit to compile a list of files that are changed with that commit, and each entry contains the number of lines of code that has been added or deleted during that change.

We made a choice to only use source code files in our analysis, and identified them with the methodology of Geominne *et al.* [23]. Code is the main material and substance of OSS and also the main focus of developers, thus we study them as to extract collaborative development. Code files also make up the roughly 90% of files in our dataset, so limiting our study to this subset will not drastically change the number of identified CoGs, but it will improve their quality. For example, there are files that are touched every time a developer submits changes, *e.g.,* "changes.txt". These would produce large CoGs for very extended durations, which is meaningless and skews our results and so they must be removed.

4.2 Merging Aliases

Merging duplicate aliases is necessary for two reasons: 1. A person identified as two or more people might be working on a single file using a number of his/her aliases, and thus could be erroneously recognized as a group; 2. A person with multiple ids might be working on two or more different files using different aliases, and thus the attribution of focus will be wrong.

Due to the small number of developers in each project (in order of tens), the merging process was done manually. All names and email IDs were checked and those with similar name, or similar email or names that highly suggest similarity were merged *e.g.,* (John Smith, smith@gmail.com), (Smith,

John@smith.com), (John S., J.smith@ucdavis.edu) are considered to be the same person.

4.3 Extracting Groups

We make the precise definition of a putative CoG by designing an algorithm which identifies sets of developers whose commits overlap in code and time. It depends on both *temporal proximity*, Δt, the maximum time separation between two consecutive CoG commits, and *code proximity*, which captures a level of modeling at which CoGs work. We reason about the appropriate practical choices for these later, after we present the generic algorithm.

1. For each code proximity, we process in increasing temporal order the commits therein, and derive a list of *contribution sequences*. Each sequence corresponds to the contributions of a set of people to that code proximity.
2. A commit is added to a sequence if it lies within a Δt time apart from the last one already in the sequence. If no prior commit lies within Δt of the current one being processed, we start a new contribution sequence with it.
3. After all commits have been processed, we remove all final contribution sequences solely comprised of commits by a single developer.
4. We also excise consecutive commits by the same person occurring at the beginning or the end. The reason is that the head or tail sequences usually represent a time of solitary development before new developers joined the existing one and formed a group. For example, the sequence

$$F_1 : P_{1,1}, P_{1,5}, P_{1,8}, P_{2,8}, P_{2,9}, P_{1,10}, P_{2,12}, P_{2,13}, P_{2,15}$$

   is reduced to:

$$F_1 : P_{1,8}, P_{2,8}, P_{2,9}, P_{1,10}, P_{2,12}.$$

   Here $F_i : \ldots$ shows the sequence of changes made to code proximity i, and $P_{j,k}$ denotes changes made by person j at time k.
5. Since developers in the same CoG may be working on more than one code proximity at the same time, *e.g.,* they may have divided tasks among each other, we merge contribution sequences of the same developer set if their lifespans are within Δt of each other, even if they belong to different code proximities. For example, if we have

$$F_1 : P_{1,8}, P_{2,8}, P_{2,9}, P_{1,10}, P_{2,12}$$

$$F_2 : P_{2,10}, P_{1,11}, P_{2,11}, P_{2,14},$$

   we merge them into:

$$F_{1,2} : P_{1,8}, P_{2,8}, P_{2,9}, P_{1,10}, P_{2,10}, P_{1,11}, P_{2,11}, P_{2,12}, P_{2,14}.$$

   We extract the putative CoGs from those last sequences.

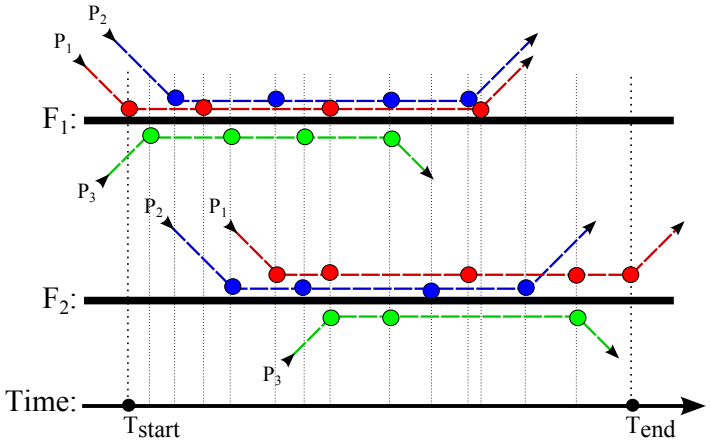

Fig. 2. An illustration of a sample CoG of size 3 (developers P_1, P_2 and P_3), and strength 2 (code proximities F_1 and F_2). The lifespan of this group is $l = T_{end} - T_{start}$.

The number of developers in each putative CoG is its "size" and the number of modules in each group is its "strength". A sample putative CoG can be seen in Figure 2. This definition of a CoG is closely related to the concepts of "implicit teams" and "succession" in Mockus' research [43].

4.4 Choices for Temporal and Code Proximity

We thought of two possible choices for code proximity, a file and a package, and based on the existing literature [33, 53], both seem to be valid choices. The most obvious choice for code proximity is that of a file, and this level of interaction is easy to mine directly from the commit logs. However, it may not provide the appropriate level of modeling putative CoGs as, often, adding a feature or fixing a bug requires touching multiple files in the same package, and even beyond. The consequence is that when developers divide tasks, they may split those files into non-overlapping groups in order to maximize throughput among them. Thus, people touching different files in the same package may still be working together. For this reason, we decide to use package-level as our code proximity. [2]

To chose the appropriate Δt we reasoned that, on average, contribution to an ASF OSS project may not be a developer's top priority, so he may want to postpone it a day or two, or even do it on weekends. On the other hand,

[2] Most of our studied projects are written in Java where files within the same file directory are considered to be in the same package. The three non-java projects, "axis2_c", "log4net", and "log4php", use the same file structure as their Java counterparts,"axis2_java" and "log4j".

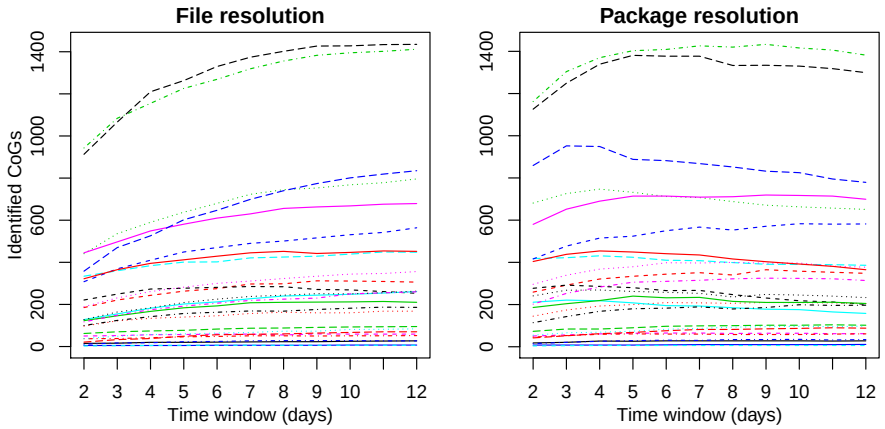

Fig. 3. The effect of time windows size on number of groups identified in projects. Each line represents one of the 26 projects.

too big a time window would result in two non-related changes to a file to be counted towards as collaboration. We chose Δt to be one week (7 days) in this study.

To make sure our choices for temporal and code proximity were sensible, we applied the above algorithm with a range of possible parameter values: file or package level for code proximity, and between 2 to 12 days for Δt. The number of CoGs identified in each project for all combinations are plotted in Figure 3. Based on the stability of the results and the above reasoning we have concluded that our choices are appropriate. [3]

We also note that our choices are also greatly in agreement with the prior study, that used slightly finer time intervals [70].

4.5 Mining Developer Mailing Lists

Mailing lists serve as the main communication hubs for OSS projects [25, 26], and have been studied in recent years [22, 71]. To evaluate whether putative CoG members coordinate among themselves, we extract messages and responses between developers from developer mailing lists.

The first challenge in mining these lists is unmasking user aliases. The large number of participants in the mailing lists makes it very difficult to manually curate the participants' aliases. Pagano and Maalej who investigated on blogs as form of coordination and communication tool used Dice similarity

[3] We also generated all the results for Tables 3, 6 and 10 for choices of 2 and 5 days for Δt. While the results were slightly different, the overall theme of the tables remained consistent with our original choice.

to merge duplicate blogs [51]. We use parsers, that we have developed previously, which employ a technique introduced by Bird *et al.* [5], which we later improved upon [22], to semi-automatically unmask aliases. The code for both parsers is available at `http://www.gharehyazie.com/supplementary/teamwork/miningscripts/`. In Appendix B we describe how we manually verified the correctness of these scripts on a sample of our data, where they exhibited 99% accuracy. These scripts have also been used in a number of previous studies [5, 21, 22, 71].

Next we describe how we construct the links between people. Messages sent to a mailing list are broadcast to all subscribed participants. Person-to-person exchanges can be inferred, using the "in-reply-to" field in the messages. When person B sends a broadcast message in reply to another broadcast message originally sent by person A to the list, we infer that B intends to communicate with A [5], and record it as a message from B to A.

4.6 Extracting Message Text

To study the content of the messages posted by developers, we first process them to eliminate artifacts. Messages usually contain the body of the message they are in response to. To remove those we filter out patterns highlighting prior text, starting each line with a special character such as '>', '|', '}' , or '*', or all the lines after tags such as: "----begin forwarded text----". We also remove punctuation and numbers, and convert all messages to lowercase letters.

We use these texts to create "word clouds". When creating word clouds from text, it is common to perform *stemming*, and to remove *stop words*. We omitted stemming because we are interested in looking at occurrences of verbs and nouns, and stemming works against this distinction. We did remove stop words, but not the *standard* English stop words. We want to see the occurrence of words such as "you", "I", and "us". Apart from these common but interesting words, other very common words such as "the", "this", and "that" were removed.

4.7 Generating Randomized Datasets

To evaluate the significance of the putative CoGs frequencies in each project, we compare their counts against a randomized baseline model, *i.e.,* chance collaboration. We generate a baseline distribution by randomizing the commit dataset for each project 20 separate times. This small number of random samples was imposed by the computational complexity of the operation and the total number of different projects; it still provides sufficient statistical power for our purposes.

Table 2. A sample changelog along with a valid and two invalid randomization examples. Sample 1 is invalid because file 3 is changed before its first change in the original data. Sample 2 is invalid because file 1 and file 3 are changed with a different frequency than original data.

Original		Valid		Invalid 1		Invalid 2	
logid	fileid	logid	fileid	logid	fileid	logid	fileid
1	1	1	1	1	1	1	2
1	2	2	2	1	3	2	3
2	1	3	1	2	2	3	4
2	3	3	2	3	1	3	1
3	4	3	4	3	4	4	1
3	1	4	1	4	2	4	2
4	1	4	4	4	1	5	1
5	2	5	1	5	1	5	3
5	4	5	3	5	4	5	2

We randomize the commit data by changing the files that a developer touches during a commit to a random set of files of equal cardinality, chosen from all available files at the time of that commit. This ensures the temporal order of the commits is preserved while the files changed during each commit are randomized. [4] We take care that files introduced later in a project are not changed before they were introduced in the project by only assigning a file to a commit after it had been introduced in the original data.

We also take care to keep the file size the same as in the empirical data. When we randomize files through commits, we preserve the number of times a file has been changed plus the number of lines added and deleted, thus file size remains similar to the original data, with respect to each time they are changed, *e.g.,* a file that has been changed three times has the same size after each commit in the randomized dataset that it has in the original dataset after the same number of commits, but *when* these changes were made may differ between the two. A sample randomization is presented in Table 2 for further clarification.

The alternative randomizations of a) shuffling people around, and b) shuffling the commit dates are not as desirable. The former randomization is not realistic, because each developer stays with a project for a limited time (around $2-3$ years). If we randomize developers across commits, it would appear that all developers are spread out thinner, in terms of commit numbers, but across the whole life span of the project (from 6 to 12+ years). The latter option is even worse. Not only it has the shortcomings of the previous alternative, but also distributes each file across the whole project time line, and it removes spikes in temporal commit activity on which the definition of group collaboration relies on.

[4] The reason that we speak of files instead of packages at this stage is that commit datasets record files, and to randomize them, we have to randomize at a file level. All results extracted from these randomized datasets are still based on package level code proximity.

4.8 Developer Survey

We designed a short survey to inform our assumptions and qualitatively verify our findings on the issue of collaborative development in ASF projects. The questions were designed to be concise and as unambiguous as possible. A copy of the questionnaire can be found in Appendix A.

When selecting subjects for this survey, and designing the questions, we faced a number of challenges. The first one was that we needed developers who had participated in a project long enough to potentially have a meaningful form o collaboration with others. For example just one month of contributions is not long enough to form a solid opinion on the state of collaboration in a project, so we selected the developers with at least 2 years of contribution experience.

The second challenge was passage of time and memory recollection. As previously mentioned, the selected projects were at least two years old at the time of data gathering (March 2012). Many developers have since left the projects, and/or may not have a clear recollection of how things were, say 10 years ago. This affects us in two major ways. First, We cannot use the survey to directly verify our results through developer confirmation. For example, we cannot ask: *"were you collaborating with person X around date Y on module Z?"* This is too specific of a question and even if people do provide an answer, we cannot be very confident in their recollection of the events. We need to design the questions towards the general act of collaboration and cooperation than any specific instance. Second, we want to ask people with the most recent experience so that their recollection will be as accurate as possible, so we chose developers who were active at the end of the data gathering.

This leaves us with 85 developers, who we contacted through email and invited them to participate in the survey. Participation was voluntary and confidential, and was expected to take less than 10 minutes. We received 9 responses to the survey, a response rate of roughly 11%. Apart from the mentioned difficulties, several factors also contribute to the low response rate, mainly the fact that the questionnaire is classified as "junk mail" by many developers. As one of the corresponding developers mentioned: "Being an active OSS developer means I get asked to fill in surveys several times a month...What is the benefit for (project name), the ASF or myself?". Taking this into account rather explains our lower than expected response rate [4]. The lesson we took from this experience was that incentivization is crucial for high survey response rates.

4.9 Measures of Focus and Ownership

To reason about the relationship between the distribution of commits across packages and group collaboration, we use the *Developer Attention Focus* (DAF) and *Module Activity Focus* (MAF) measures [54]. DAF measures a developer's *focus*, *i.e.,* if they concentrate their efforts on very few files rather than work-

ing on many and MAF measure *ownership*, *i.e.,* whether there is just one or very few developers who wrote most of that mentioned file or package. These metrics were initially introduced by Bluthgen *et al.* [8] and here we give a brief description of them for completeness.

Let $W_{n,m} = \{w_{i,j}\}$ denote a commit matrix for a project of m developers and n modules, where $w_{i,j}$ denotes the number of commits by developer j to modules i. Then the total commits by developer j is $D_j = \sum_{i=1}^{n} w_{i,j}$ and the total commits that a module i receives is $M_i = \sum_{j=1}^{m} w_{i,j}$. Here *module* can be a file or a package. We use the latter in our measurements to be consistent with our definition of collaboration. The total commits to all modules is then $A = \sum_{i=1}^{n} \sum_{j=1}^{m} w_{i,j}$, the proportion of commits to module i is $r_i = M_i/A$, and the proportion of commits by developer j is $q_j = D_j/A$. The proportion of commits by developer j to module i is $q'_{i,j} = w_{i,j}/D_j$ and the proportion of commits to module i is $r'_{i,j} = w_{i,j}/M_i$.

Based on these definitions, the DAF and MAF metrics are given by:

$$\mathcal{DAF}_j = (\delta_j - \delta_{j_{min}})/(\delta_{j_{max}} - \delta_{j_{min}}),$$

and

$$\mathcal{MAF}_i = (\delta_i - \delta_{i_{min}})/(\delta_{i_{max}} - \delta_{i_{min}})$$

where [54]:

$$\delta_j = \sum_{i=1}^{n} \left(q'_{ij} \ln \frac{q'_{ij}}{r_i} \right),$$

and

$$\delta_i = \sum_{j=1}^{m} \left(r'_{ij} \ln \frac{r'_{ij}}{q_j} \right).$$

Based on these definitions, DAF and MAF range from 0 to 1. A low DAF (closer to 0) means that a developer has spread his commits across many files rather evenly, and a high DAF (closer to 1) means that all of their commits concern very few files. Similarly a low MAF would mean, that commits on a file come from many users thus the file has a low ownership, while a high MAF means most commits on that file are coming very few developers, *i.e.,* the file has a high ownership and those developers are called the *file owners*.

These measures of focus are based on cross entropy and can also be seen as related to entropy based inequality indices such as the Theil index used to aggregate software metrics [62, 64].

5 Results and Discussion

5.1 RQ1: Identification, Validation and Verification of CoGs

Using the algorithm above we identified putative CoGs in each of the 26 OSS projects described above. Table 3 gives the number of CoGs in each project

Table 3. The number of identified and confirmed CoGs in each project, along with minimum, average and maximum number of CoGs in the randomized dataset.

	Collaborative Groups		Randomized Data		
	Identified	Confirmed	Min	Average	Max
abdera	28	14	35	41.5	48
activemq	351	161	1492	1544.0	1589
ant	707	299	2296	2335.8	2399
avro	30	24	112	127.4	141
axis2_c	408	129	780	813.8	841
camel	709	325	1952	2015.8	2066
cassandra	267	32	308	327.2	344
cayenne	208	43	356	380.9	406
cxf	1426	291	6980	7054.1	7126
derby	868	521	2101	2166.7	2226
hadoop_hdfs	193	35	316	340.0	362
harmony	65	17	125	133.2	143
hive	253	58	394	411.9	432
ivy	60	11	77	83.7	89
log4j	99	28	226	236.4	256
log4net	10	4	16	17.8	20
log4php	6	5	13	15.3	18
lucene	397	223	1105	1140.0	1171
mahout	189	111	535	556.8	577
nutch	82	41	195	203.2	214
ode	234	99	500	523.1	541
openejb	567	132	3327	3369.2	3413
pluto	87	45	223	230.5	239
solr	315	159	849	865.5	907
wicket	1377	220	2792	2840.1	2892
xerces2_j	435	20	1041	1076.1	1109

along with the average, min and max number of CoGs in the randomized dataset. We also generate a normal distribution from the randomized datasets' number of identified CoGs and calculate the probability that the empirical results belong to the same distribution *i.e.,* the probability of the number of identified CoGs, given the normal distribution. The probability in all cases is smaller than 1%. The randomized models are meant not as a model of an actual developer collaboration, but rather as a way of excluding the most obvious random behavior.

We illustrate next the distribution of group sizes within and between projects. For two randomly selected projects, we have provided box-plots of the distribution of group sizes in our baseline random models in Figure 4. The red line shows the distribution of group sizes in our empirical data. The empirical data seems to follow the same distribution, but at significantly lower levels.Groups above a certain size do not occur in our empirical results.

The above results indicate that the frequency of identified collaboration is significantly lower than expected, i.e., as compared to levels predicted by the randomized models. This is consistent with developers' being selective about

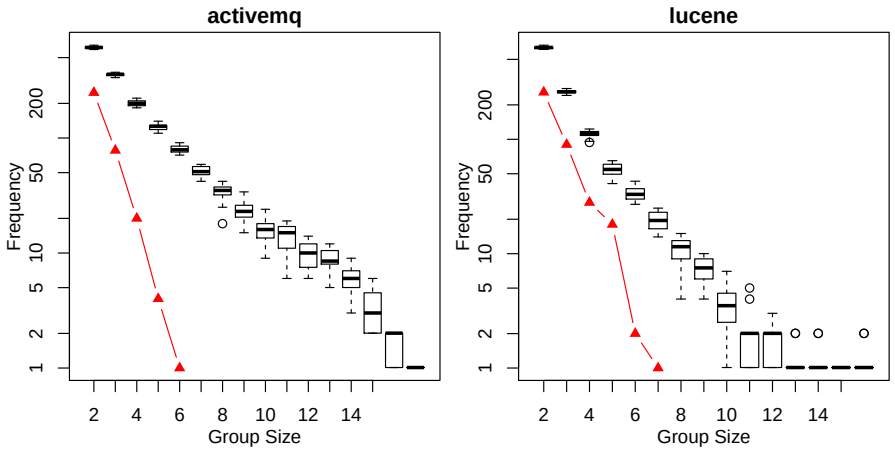

Fig. 4. Distribution of CoGs' size in two OSS projects. The box-plots show the distribution of the randomized baseline models and the red lines shows the empirical results.

collaborations or working on a team. It also is consistent with the notion that randomly forming teams may not be beneficial. In addition, our findings that larger groups are much less prevalent than smaller ones, is arguably what we expected as the costs of coordination and communication increase with group size, along with the likelihood of merge-conflicts and other such difficulties.

5.1.1 File Name Mentions as Evidence in Support of CoGs as Teams

To further assess whether the extracted putative CoGs are meaningful, we look at the content of messages exchanged between CoG members.[5] We mined the *developer mailing list* archives for messages between developers, as described. For each putative CoG, we searched in all messages sent between developers for the names of files from their contribution sequences. [6] A putative CoG is considered *confirmed* if the exact name of at least one of those files is found in the subject or body of the message, during that CoG's lifespan. The results from our approach are presented in Table 3, in the confirmed column. We managed to identify at least one of the files a putative CoG was inferred to have been working on in 38% of CoGs in all projects. This number was (not unexpectedly) much lower, around 16% on average, for projects where com-

[5] We scanned by hand a number of CoGs and were able to identify via the contents of their messages that *developers were truly coordinating their collaboration* as predicted. That encouraged us to come up with the automated, but necessarily more simplistic, large-scale analysis, presented here.

[6] We search for files within the packages subject to collaboration since in technical discussions, file names occur naturally and more frequently than package names.

Table 4. The number of identified and confirmed CoGs in each project. This is similar to Table 3 but only done with CoGs extracted with $\Delta t = 2$ and $\Delta t = 5$.

	Collaborative Groups $\Delta t = 2$		Collaborative Groups $\Delta t = 5$	
	Identified	Confirmed	Identified	Confirmed
abdera	18	8	26	14
activemq	259	80	334	137
ant	681	217	732	284
avro	15	8	29	20
axis2_c	418	83	424	111
camel	580	266	714	333
cassandra	277	13	280	25
cayenne	145	21	200	38
cxf	1162	149	1403	252
derby	859	444	888	519
hadoop_hdfs	211	34	208	40
harmony	51	15	64	17
hive	241	46	262	59
ivy	46	5	61	10
log4j	73	15	90	26
log4net	8	4	8	4
log4php	4	4	6	5
lucene	296	161	379	203
mahout	114	53	180	94
nutch	42	20	68	35
ode	186	39	240	79
openejb	416	64	524	102
pluto	55	22	74	37
solr	202	92	306	143
wicket	1126	139	1381	195
xerces2_j	404	12	449	18

munications level was much lower than development activity *i.e.,* there were fewer messages in their mailing lists than commits in the repository. Interestingly, when we examine teams identified with a smaller *temporal proximity*, the percentage of confirmed teams drops to 30% for $\Delta t = 5$ and 26% for $\Delta t = 2$ (see Table 4). This further strengthens our confidence in our choice of *temporal proximity* for CoGs.

This approach likely underestimates the number of CoGs that coordinate via the mailing lists as it is possible that even though group members are communicating, file names may not be mentioned, or may be mentioned in an inexact form. It is also likely that they use other communication methods, as pointed out by developers in our survey, *e.g.,* IRC or personal email for coordination, even though it is discouraged by ASF.

Taking the underestimation into account, the confirmed numbers are strong evidence supportive of the hypothesis that a high number of CoGs coordinate and collaborate explicitly. This validates, in part, our algorithmic approach for identifying CoGs, as it shows that when we identify a CoG it is likely that it is a real team.

Fig. 5. Difference word cloud of words used within messages within and outside of CoGs. The size of each word represents the extra prevalence it has in the corresponding group.

5.1.2 Language Differences in Group vs. Solo Mode Communication

We were also interested to learn whether there is any semantic difference between those messages posted while developers were involved in CoGs and while they were working solo. As previously mentioned, we extracted the text of each message and parsed their words. From them, we created two distributions of word frequencies: "group" word distribution, of all words used in messages posted by people during their time spent in CoGs, and "solo" word distribution, of all words used in messages posted by people during their time spent not in CoGs. Each distribution was normalized by the number of messages in each group. A comparison contrasting word cloud of the resulting word distributions is shown in Figure 5, where we see a stark difference between group-mode language and solo-mode language. [7]

We observe that while in group-mode, developers use more action words, such as "should", "change", "think", and "release", along with task oriented

[7] The word cloud was created using the "comparison.wordcloud" function in the "wordcloud" package in R.

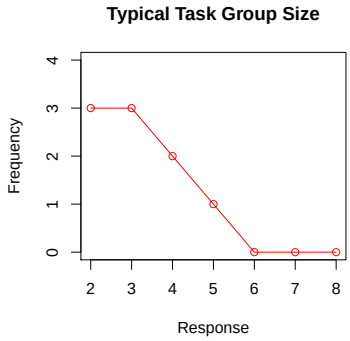

Fig. 6. The frequency of responses to the typical number of collaborators on a task.

terms like "task", "work", "add" and "remove". These words are in great agreement with commit message language found by Maalej and Happel [40,41], which together suggest that CoG conversations may be more task oriented than solo developer messages.

In contrast, solo-mode messages are more ripe with words like "bug", "debug", "build", "trace" and "exception", hinting at greater focus on removing defects and improving quality rather than adding new features to the software. We also point out the prevalence of negative terms such as "don't" and "doesn't" in group messages versus solo-mode messages.

Thus, the group work conversation in CoGs seems different than the conversation that individuals not in CoGs are having, with the latter having less indicators of coordination than the former. This is consistent with people's behavior in teams, when they communicate to coordinate and exchange information, versus requesting help or reporting issues, typically done while working individually.

5.1.3 Survey Results

The survey of the ASF developers offers means for triangulation of our results. The developers who responded largely agreed with each other on most topics. Here we discuss the findings from our survey as they pertain to this RQ; others will be presented with the subsequent RQs. According to our survey results presented in Table 5:

Working on a project is a *collaborative* effort: 8 out of 9 Participants agreed or strongly agreed that OSS development is a collaborative effort rather than a quilt of solitary contributions. The one person who *strongly disagreed* with this, responded *"I tended to find that only implementing new features was collaborative..."* when asked *which of their tasks were more collaborative.*

Table 5. Distribution of responses to agreement questions in the survey.

Question	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Working on the project was a collaborative effort	1	0	0	2	6
You actively attempted to "team up" with others to complete tasks	0	1	2	2	4
Collaboration increases productivity	0	0	1	3	5
Collaboration requires extra coordination and communication	1	1	1	4	2

Developers *choose* what to work on: When we asked developers about how they choose and prioritize their tasks and if they choose what they work on, we got responses such as *"Choose my own tasks, whatever itches I want to scratch"*, *"I've usually selected tasks with which I was comfortable implementing them..."*, *"what ever needs to be done"*, *"...each worked in the area they were most comfortable with and also addressed issues that were critical..."*, and *"...I jump on any issue that affects reliability or correctness."*, all indicating that it was the developer's decision to work on a specific topic and region of the project, even if this decision was based on certain criteria and priorities. Thus, working on the same code proximity is by choice, not forced. **Developers *actively* team-up to complete tasks:** 6 out of 9 agreed or strongly agreed while 1 person disagreed with this, which implies that collaboration is explicit and what we find is not just an artificial artifact of our methodology.

We asked developers about the typical time to complete a task and most of them (6 out of 9) responded: **Completing typical tasks takes** $3-5$ **days**, and two others said $1-2$ days. The median lifespan of a CoG across all 26 projects is 6 days. This shows that identified CoGs more or less correspond to actual development tasks. Figure 6 also shows the response to **the typical number of people involved in a task ranges between** 2 **and** 5 with a downwards trend towards larger teams that highly matches the range of identified CoGs size as previously discussed Figure 4.

While small, this survey was informative. The developers' answers clearly indicate their preference for choosing one's task in OSS projects, and that teaming up is task-based and organic. The surey also pointed out the preference among developers for small teams. The patterns in these answers are largely in agreement with our quantitative findings. We also noted divergent responses, reminding us of the diversity among OSS developers and their mindsets.

Table 6. Amount of person-days and commits in each project, separated by group vs. solitary development. Bold items in columns 3 and 6 represent values greater than 0.7 (significant values). Projects highlighted in the last column express greater commits per day during group vs. solitary development. CoGs are extracted with $\Delta t = 7$.

	Person Days (D)			Commits (C)			C/D	
	CoG	Solo	G/S	CoG	Solo	G/S	CoG	Solo
abdera	292	3912	0.07	358	1279	0.28	1.23	0.33
activemq	4670	17397	0.27	2855	4267	0.67	0.61	0.25
ant	13754	30986	0.44	6898	10532	0.65	0.50	0.34
avro	320	3647	0.09	169	973	0.17	0.53	0.27
axis2_c	5091	8210	0.62	4453	3372	**1.32**	0.87	0.41
camel	8955	12181	**0.74**	7008	8704	**0.81**	0.78	0.71
cassandra	4342	467	**9.30**	4155	4157	**1.00**	0.96	*8.90**
cayenne	2504	15565	0.16	2442	6241	0.39	0.98	0.40
cxf	10936	22971	0.48	7846	11265	**0.70**	0.72	0.49
derby	15227	13717	**1.11**	4979	5667	**0.88**	0.33	*0.41**
hadoop_hdfs	4339	5322	**0.82**	1068	1486	**0.72**	0.25	*0.28**
harmony	520	4976	0.10	437	1025	0.43	0.84	0.21
hive	4834	2067	**2.34**	1175	1559	**0.75**	0.24	*0.75**
ivy	876	5173	0.17	581	2141	0.27	0.66	0.41
log4j	1414	17639	0.08	1044	2752	0.38	0.74	0.16
log4net	66	4937	0.01	42	653	0.06	0.64	0.13
log4php	92	3115	0.03	176	695	0.25	1.91	0.22
lucene	5700	23342	0.24	2574	3955	0.65	0.45	0.17
mahout	1992	8058	0.25	1013	1737	0.58	0.51	0.22
nutch	884	12251	0.07	371	1466	0.25	0.42	0.12
ode	2753	6359	0.43	2309	3465	0.67	0.84	0.54
openejb	7787	29454	0.26	7646	12217	0.63	0.98	0.41
pluto	934	11724	0.08	660	1792	0.37	0.71	0.15
solr	3595	9958	0.36	1498	2571	0.58	0.42	0.26
wicket	13680	10972	**1.25**	15837	12140	**1.30**	1.16	1.11
xerces2_j	6523	16421	0.40	3522	4540	**0.78**	0.54	0.28

Result 1: *Collaborative Groups form within ASF OSS projects organically and are unlikely to be simply an artifact of our approach. CoG members communicate and coordinate through the developer mailing lists, and their communication suggests a meaningful difference between development and coordination patterns during and outside putative CoGs. The developers surveyed describe collaboration mechanisms consistent with our findings.*

5.2 RQ2: Collaboration Prevalence

To contrast the time spent and code commits submitted in each project during collaboration periods as compared to periods of solitary work, we count

Table 7. Amount of person-days and commits in each project, separated by group vs. solitary development. This is similar to Table 6, only done with CoGs extracted with $\Delta t = 2$.

	Person Days (D)			Commits (C)			C/D	
	CoG	Solo	G/S	CoG	Solo	G/S	CoG	Solo
abdera	82	3394	0.02	180	1379	0.13	2.20	0.41
activemq	1178	20188	0.06	1485	5253	0.28	1.26	0.26
ant	2746	41994	0.07	4139	12384	0.33	1.51	0.29
avro	35	3335	0.01	51	978	0.05	1.46	0.29
axis2_c	1699	11602	0.15	3315	4614	**0.72**	1.95	0.40
camel	2850	18286	0.16	4043	10591	0.38	1.42	0.58
cassandra	1603	3183	0.50	2600	5035	0.52	1.62	1.58
cayenne	523	17138	0.03	986	7051	0.14	1.89	0.41
cxf	3467	30325	0.11	4940	13092	0.38	1.42	0.43
derby	2737	24812	0.11	3249	6843	0.47	1.19	0.28
hadoop_hdfs	626	9035	0.07	835	1500	0.56	1.33	0.17
harmony	200	4391	0.05	291	1154	0.25	1.46	0.26
hive	926	5975	0.15	825	1642	0.50	0.89	0.27
ivy	141	5908	0.02	248	2265	0.11	1.76	0.38
log4j	279	18770	0.01	582	3003	0.19	2.09	0.16
log4net	20	4983	0.00	29	664	0.04	1.45	0.13
log4php	20	1967	0.01	97	736	0.13	4.85	0.37
lucene	908	27707	0.03	1444	4638	0.31	1.59	0.17
mahout	399	9651	0.04	521	2041	0.26	1.31	0.21
nutch	99	13036	0.01	132	1557	0.08	1.33	0.12
ode	622	7855	0.08	1283	4162	0.31	2.06	0.53
openejb	1580	35576	0.04	4435	14409	0.31	2.81	0.41
pluto	228	12429	0.02	325	1998	0.16	1.43	0.16
solr	613	12610	0.05	828	2952	0.28	1.35	0.23
wicket	4483	20169	0.22	10669	16795	0.64	2.38	0.83
xerces2_j	1578	21366	0.07	2277	5595	0.41	1.44	0.26

the number of commits and days that each developer contributed/spent while inside a CoG and compare those to the commits and days that developer contributed/spent while developing individually. Then, we aggregate the results for each project. This gives us a glimpse into the amount of time spent in each project by all developers, (Table 6, Columns 1 to 3), and the amount of commits contributed in each project, by all developers, (Table 6, Columns 4 to 6). There we see that only in 6 out of 26 projects, on average, developers spend a comparable ($> 70\%$) or significantly higher amount of time working in groups than working alone. And in 9 out of 26 projects, on average, there are comparable ($> 70\%$) commits during group collaboration than solitary work. From the above, we can also measure the number of commits per unit time, and compare them during periods of collaboration and periods of personal work, in Table 6, Columns 7 and 8. Interestingly, while most projects expressed a smaller portion of commits and person days assigned to groups, the ratio of commits/(person days) is greater in CoGs in all but 4 of the projects. Thus, we observe that although the amount of time and effort spent during collaboration differs from one project to another, and in most cases it is not a big

Table 8. Amount of person-days and commits in each project, separated by group vs. solitary development. This is similar to Table 6, only done with CoGs extracted with $\Delta t = 5$.

	Person Days (D)			Commits (C)			C/D	
	CoG	Solo	G/S	CoG	Solo	G/S	CoG	Solo
abdera	212	3992	0.05	287	1310	0.22	1.35	0.33
activemq	3219	18848	0.17	2479	4578	0.54	0.77	0.24
ant	9071	35669	0.25	6146	11042	0.56	0.68	0.31
avro	208	3759	0.06	140	975	0.14	0.67	0.26
axis2_c	3815	9486	0.40	4101	3776	**1.09**	1.07	0.40
camel	6410	14726	0.44	6271	9258	0.68	0.98	0.63
cassandra	3536	1273	**2.78**	3919	4361	**0.90**	1.11	*3.43**
cayenne	1586	16418	0.10	1922	6538	0.29	1.21	0.40
cxf	8369	25472	0.33	6985	11821	0.59	0.83	0.46
derby	11121	17823	0.62	4604	5996	**0.77**	0.41	0.34
hadoop_hdfs	2823	6838	0.41	1027	1488	0.69	0.36	0.22
harmony	405	5091	0.08	409	1046	0.39	1.01	0.21
hive	3223	3678	**0.88**	1096	1576	**0.70**	0.34	*0.43**
ivy	481	5568	0.09	459	2190	0.21	0.95	0.39
log4j	811	18242	0.04	871	2845	0.31	1.07	0.16
log4net	35	4968	0.01	35	658	0.05	1.00	0.13
log4php	92	3115	0.03	170	699	0.24	1.85	0.22
lucene	3318	25723	0.13	2271	4146	0.55	0.68	0.16
mahout	1299	8751	0.15	894	1829	0.49	0.69	0.21
nutch	502	12633	0.04	291	1501	0.19	0.58	0.12
ode	1775	6760	0.26	1963	3697	0.53	1.11	0.55
openejb	5324	31917	0.17	6795	12817	0.53	1.28	0.40
pluto	638	12020	0.05	547	1874	0.29	0.86	0.16
solr	2155	11398	0.19	1325	2674	0.50	0.61	0.23
wicket	11006	13646	**0.81**	14700	13268	1.11	1.34	0.97
xerces2_j	4327	18617	0.23	3168	4865	0.65	0.73	0.26

portion, there seems to be an increase in productivity during these collaborations. This difference between productivity within a CoG against individual development exists and is even stronger when we lower the *temporal proximity parameter* of our CoG extraction mechanism (Tables 7 and 8).

To study whether tenure, or time spent with the project, affects developers' inclination towards co-development, we select developers with at least 2 years of commit activity (the dates of their commits span more than 2 years). We call them *dedicated* developers. We then measure the number of groups each of these developers participated in co-development during their first and last years as developers. We also count the number of days they spent in CoGs during the first and last years. The results in Table 9 show that over all projects, most developers ($148/214 = 69\%$) participate in more groups in their first year. We also see that over all projects, the difference in the days participating in CoGs in the first vs. the last year ($129/214 = 60\%$) is slightly above indifference (50%). This suggests that developers are not less inclined to collaborate later on, but instead they participate in slightly fewer, longer lasting groups. One explanation for this is that, perhaps, the eagerness of youth

Table 9. The correlation between tenure and co-development. "Group Count" columns "First" and "Last" show the number of CoGs a dedicated developer has participated in during their first and last developer years. "F>L" shows the number of dedicated developers who have participated in more CoGs in their first year than their last."Group Days" columns are similar, but show the days a dedicated developer has been part of a CoG.

	Developers		Group Count			Group Days		
	All	Dedicated	First	Last	F>L	First	Last	F>L
abdera	7	2	18	0	2	729	0	2
activemq	25	13	300	127	7	4032	3899	6
ant	37	18	465	47	15	5636	3797	13
avro	10	1	14	9	1	365	308	1
axis2_c	20	10	450	128	8	3363	3311	4
camel	30	13	546	414	10	4857	4518	6
cassandra	13	2	83	170	0	734	734	1
cayenne	18	10	154	29	9	3381	1139	10
cxf	38	18	1160	587	13	6366	6310	13
derby	29	15	580	386	11	4599	5170	7
hadoop_hdfs	23	1	15	8	1	336	381	0
harmony	16	3	0	53	0	0	869	0
hive	18	2	91	99	1	749	731	2
ivy	6	4	33	15	2	1033	836	3
log4j	13	9	88	41	6	2830	1381	7
log4net	5	3	4	2	1	852	672	1
log4php	6	2	4	3	1	730	623	2
lucene	32	15	199	147	9	4977	4460	4
mahout	13	5	51	90	2	1558	1743	1
nutch	16	8	61	19	6	2731	1888	6
ode	15	3	134	65	3	1017	727	1
openejb	35	17	233	225	8	5495	4105	11
pluto	21	6	39	34	4	1794	1523	4
solr	18	9	188	112	7	2900	2358	5
wicket	24	12	1049	248	11	4779	4169	9
xerces2_j	26	13	379	173	10	4790	4168	10
Total	514	214	6338	3231	148	70633	59820	129

over time gets exchanged for higher focus, or maybe loyalty, in latter years. Our findings also point to another interesting and expected phenomenon: that people tend to build lasting relationships, which survive best in smaller groups. From a social perspective, it could be argued that this is consitent to the tendency of most people to belong to a smaller, more personal community, within larger projects or society. Such smaller, more persistent groups can cut down coordination and increase trust.

Result 2: *Developers spend a considerable fraction of their development time working in groups, and while their patterns of contribution and effort within CoGs vary over projects, in aggregate when in CoGs, they submit more code per unit of time. In the developers' latter time with a project they collaborate for almost the same amount of time, but participate in fewer groups.*

These results also have implications for practitioners as they can point them towards more efficient and less distracting coordination practices. Namely, working in a group is to be expected, and teams should be chosen well and social links nurtured. As experience is gained, spending more time in a team or task can be more rewarding. Developers with more experience can concievably be resources for information on good teams and how to make the best out of one's time in OSS projects.

5.3 RQ3: Focus vs. Collaboration

To study the correlation between group collaboration and developer and package focus, we measure the focus of all developers and on all packages in each project.

First, we investigate the correlation between developer focus and CoG membership. Ideally, to compare more and less focused developers, we would select from the first and last quartiles in each project. However, since many of the projects have a small number of members, these quartiles would have 3 or fewer members, thereby precluding meaningful statistical analyses. Instead, we pool over all projects, *i.e.,* we take the first, Q_1, and last, Q_4, quartiles, with respect to focus, of developers from each project, and we merge the developers from all projects into two quartile lists. Thus, the first group are developers whose DAF is less than 25% of the population in their project, *i.e.,* the less focused group, and the second group are those whose DAF is greater than 75% of their projects' population, *i.e.,* the focused group.

For each group we measure the number of days the developers were part of a putative CoG. A beanplot [35] of the distribution of group membership days for these groups is shown in Figure 7. The data shows that the number of days spent in putative CoGs is negatively correlated with developer focus, *i.e.,* the more focused one is, the less likely one is to be part of a group, and vice versa. A *Mann-Whitney U test* of these two populations shows there is a statistical difference between the two (with a p-value of $< 10^{-3}$) and that we are not mislead by their visual difference.

We repeat the same process for packages, separating them and then merging them into two groups: the first group is made up of packages with MAF less than 25% of packages in their project and the second, packages with MAF

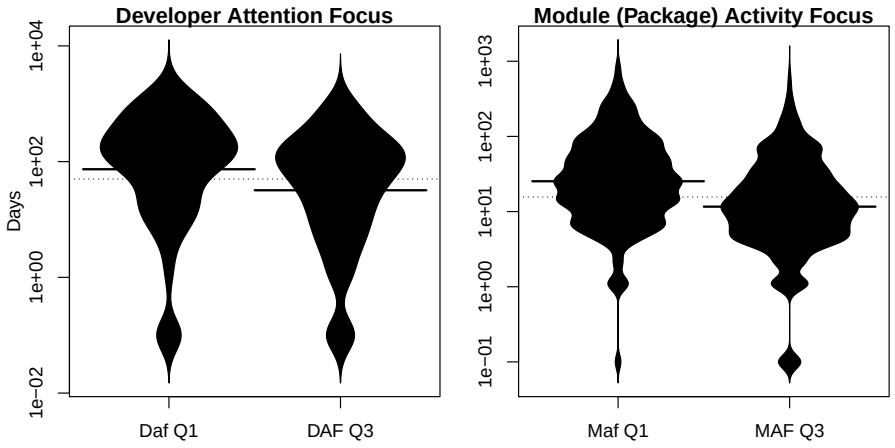

Fig. 7. Distribution of collaboration time vs. focus. Left: Distribution of collaboration days for less focused vs. focused developers. Right: Distribution of days involved in collaboration for packages with high vs. low ownership.

greater than 75% of packages within their project. A beanplot of the distribution shows that there is also a negative effect between MAF and being in a putative CoG, *i.e.,* the higher a package's attention focus, the less likely it is being collaborated on in a group. We also confirmed this difference by running a similar Mann-Whitney U test on package focus groups (p-value $< 10^{-10}$).

Result 3: *Developers with more focus collaborate less often, and packages with higher ownership are also less likely to be subject to collaboration.*

When it comes down to atomic programming tasks, be it an algorithm implementation, or coding up a web page, it is reasonable that one person would take on the lion's share of the development, and files containing such tasks won't be collaborative venues. This, in conjuction with the previous results, indicates that different programming activities may be in varying need of collaborative attention, *e.g.,* adding features may need more collaboration than a well developed algorithm implementation. Mapping such tasks onto individual programmer's needs, abilities and available time can potentially aid in the self matching to the available tasks.

5.4 RQ4: Productivity and Effort vs. Collaboration

We study productivity using two of the metrics previously used by Xuan and Filkov [70]; code effort and code growth. Code effort is defined as the number of lines added plus number of lines deleted from each package, at each commit.

Table 10. Correlation between co-development and developer productivity. The first 4 columns shows the difference between group and solitary development in terms of LOC per commit per file. Values greater than 3 lines have been highlighted as a significant difference. The values are rounded, so a negative 0 means the changes were negative, but close to zero. Empty cell demonstrate lack of statistical significance.

	Add	Del	Effort	Grow	Age	Msgs.	Files	Cmts.	Devs.
abdera	-0	0	-2	-2	2111	2921	3193	1492	13
activemq	-1	1	**-25**	-2	2283	20896	16788	6124	28
ant		0	**-9**	-0	4447	20433	11620	14710	44
avro	-1	0	2	-3	1074	7114	3021	984	12
axis2_c	**-6**	1	**-18**	**-5**	2995	15201	10262	6742	24
camel	**-28**	1	**-25**	**-33**	1749	27431	36965	12257	31
cassandra	**-46**	1	**-52**	**-55**	1121	4105	17125	5968	13
cayenne	-0	0	**-15**	-2	2189	6247	31489	7514	20
cxf	3	-0	1	**11**	2052	7952	37867	15322	45
derby	-0	-0	-1	0	2781	74850	6563	8301	35
hadoop_hdfs	-2	1	**-10**	-3	998	2608	1153	1529	25
harmony	**-39**	0	**-39**	**-48**	2378	31855	14898	1377	25
hive	-3	1	**-4**	**-5**	528	11373	7333	1715	18
ivy	1	0			397	907	3513	2347	9
log4j	-0	1	**-8**	-1	4114	5826	5519	3377	18
log4net					2975	1199	1060	683	7
log4php	-2	1	-3	**-11**	2263	1175	1409	803	9
lucene	-0	-0	**-28**	-1	3846	66817	6674	5343	41
mahout	-2	1	-3	**-4**	1519	18640	5123	2280	15
nutch	**-5**	2	**-4**	**-15**	2594	13041	3072	1600	16
ode	**-4**	1	**-8**	**-10**	2225	7823	11006	4901	17
openejb	1	-0	**-23**	**13**	2080	8191	43960	16620	38
pluto	-0	0	**-6**	-0	2907	4038	5971	2150	24
solr	-2	0	**-17**	**-6**	1548	21359	8534	3354	19
wicket	0		**-17**	0	1999	12479	48045	24059	24
xerces2_j	1		2	1	4373	4744	3732	7336	33

Code growth is number of lines added minus number of lines deleted from each package, at each commit, *i.e.,* it shows a package's size change in terms of LOC per commit.

In each commit a certain number of packages are changed; some number of lines are added and some are deleted. We group commits into those during group collaboration and those during personal development but we only count commits from developers that at some point are part of a CoG, as we wish to measure the difference in their productivity during and outside collaborations. We then measure code effort and growth on all developers in each project, for both groups of commits. If there is a meaningful statistical difference between the two populations, then developer productivity has changed during group collaboration. The results are presented in Table 10. The values in columns $1-4$ represent the confidence interval of the *Mann-Whitney U test* of the two

Table 11. Correlation between co-development and developer productivity. This table is similar to Table 10, only for $\Delta t = 2$ and $\Delta t = 5$.

	$\Delta t = 2$ Days				$\Delta t = 5$ Days			
	Add	Del	Effort	Grow	Add	Del	Effort	Grow
abdera	-1	0	-3	**-4**	-1	0	-3	-2
activemq	-0	1	**-20**	-1	-0	0	**-22**	-1
ant		0	-2	0		0	**-5**	
avro	-2	1	**-6**	**-9**		0		
axis2_c	-1	0	**-5**	-0	**-4**	1	**-11**	-2
camel	**-21**	1	**-19**	**-28**	**-26**	1	**-23**	**-32**
cassandra	**-24**	1	**-34**	**-30**	**-40**	1	**-47**	**-50**
cayenne	**-3**	-0	**-27**	**-5**		-0	**-10**	
cxf	**8**	-0	2	**22**	3	-0	2	**13**
derby	-0	-0	-1	0	-0	-0	-0	0
hadoop_hdfs	-1	0	-3	-1	-2	1	**-6**	-2
harmony	**-49**	1	**-50**	**-58**	**-34**	0	**-36**	**-42**
hive	-0	1	-1	-2	-2	1	**-3**	**-4**
ivy	2	0	1		1	1		
log4j		1	**-5**	-0	-0	1	**-8**	-1
log4net							-3	
log4php	-1	2		**-4**	-2	1	-3	**-11**
lucene	-1	-0	**-17**	-1	-0	-0	**-21**	-0
mahout	-2	1	**-3**	-3	-1	2	-2	-3
nutch	**-4**	1	**-5**	**-8**	**-5**	2	-3	**-16**
ode	**-5**	1	**-6**	**-14**	-3	1	**-7**	**-10**
openejb	1	-0	**-17**	**8**	0	-0	**-22**	**4**
pluto	1	-0	-2	1	-0	0	**-5**	-0
solr	**-8**	1	**-25**	**-14**	-1	0	**-15**	-1
wicket	0	-0	**-17**	2	0	-0	**-18**	2
xerces2_j	0	-0	4	1	1	-0	3	2

populations. The numbers are rounded into integers and values highlighted are the ones where the test returned a p-value of < 0.05.

We observe that in terms of code effort, in 17 projects developers showed a statistically meaningful decrease in code effort while within putative CoGs. The other projects developers' change was either statistically insignificant, or small in magnitude (3 or fewer lines of code). At the same time, code growth decreases among 10 projects' developers, but also increases among 2 projects' developers. Table 11 again shows that these results are rather stable and do not change drastically with alteration of CoG parameters.

These results are seemingly at variance with the recent results of Xuan and Filkov [70], where the code growth was found to be positive during collaboration, for groups of size 2. Cross-checking with their results, we see that our results are in fact not in disagreement. They have studied code growth in 6 ASF projects, 5 of which are present in our current study. These 5 projects are "ant", "cxf", "derby", "lucene", "openejb". What they report in terms of code growth during pair-wise co-development is compatible with our presented findings for 2 of those 5 projects as we find that same positive growth in "cxf" and "openejb". For the other 3 projects, growth was positive in 2 of them, and

insignificant in "lucene". Fig. 1 in their paper shows that the positive growth in these two projects is much less than the growth in "cxf" and "openejb". We note that one of the criteria for their selection of projects to study was to have a high number of developers, which may have an effect on group collaboration patterns, and may have contributed to their results and overall conclusion. Here, our experiments were much more comprehensive and less biased, and hence, we assert, are more conclusive.

In Table 10 we have presented a number of additional overall characteristics for all our projects. The Spearman correlation between these characteristics and code effort shows there is a significant negative correlation between code effort and a project's total number of files (-0.57). We also observe significant correlations between code growth and the number of commits and developers in a project (0.53, 0.57). These numbers hint at having more developers and activity in projects correlates positively with higher developer productivity during collaborations, while having more packages in a project may lower code effort.

5.4.1 Adjusting to Collaboration

Our results provide compelling evidence for the hypothesis that during group collaboration developers commit more often, but in smaller amounts (see Tables 6 and 10).

A plausible explanation is that smaller contributions may decrease the chances of conflicts when more people are around. We asked developers in our survey *if and how they adjust their working style when collaborating* and the most common response was that they *commit more often and in smaller sizes*, thus in agreement with our findings. This is also consistent with the author's experience in collaborative development.

While survey participants collectively believe that collaboration *increases* productivity, they also admit that *collaboration requires extra communication and coordination* (Table 5). Our results, that show collaborative groups being statistically unfavorable are consistent with this, and are possibly capturing the hidden costs and overhead of socio-technical collaboration.

> **Result 4:** *Effort spent on each package, at each commit, drops during group collaboration in most projects, and increases in only a few. Increase in code growth is correlated to project's total number of commits and developers. Committing more frequently but in smaller chunks is one of developers' methods of adjusting to a collaborative environment.*

To practitioners, this together with the earlier results, can reafirm the notion that teams take effort to form, maintain, and get the best out of. These results offer evidence towards collaboration being recognized as coming at a cost, but is generally beneficial towards lowering developer effort. In other

words, working in groups has an increased cost, but pays dividends. Thus, the cost of scaling collaborations should be an important consideration when a developer joins new projects or teams, as it may ultimately be counterproductive for larger groups. This of course is a generic issue, very well known in management science for centrally managed teams, where dealing with team scalability and corrections vis-a-vis project management is perhaps more direct [9].

6 Threats to validity

There are a number of threats to our approaches and our conclusions. First and foremost is the threat to *construct validity.* our method for recognizing groups is clearly only a second hand approximation. We argue, that if groups in fact exist, then our method must capture them, so long as they produce code at the same time. Clearly, a group of people can do work together which does not involve all of them coding at the same time. But we do not presume to be able to identify such groups, only their subsets involved in coding. The non-coding members of the group are not directly relevant to our study, so our groups may be only subsets of actual groups. Clearly, intent cannot be established from trace data, so any results must necessarily be possibly circumstantial. But by studying group prevalence over a large set of OSS projects, we hope that there will be sufficient evidence to make our results convincing. While unmasking user aliases, we used similarity between names and email addresses as a heuristic for identifying people with multiple accounts. While this is a very reasonable assumption, there is a slight chance of false positives from people with identical or highly similar names.

We also recognize several other threats to the *internal validity* of our work. We used a survey as a more direct approach of establishing the existence of groups, but passing of time limited our ability to explicitly ask specifics of the team members, forcing us to settle for less precise questions and fewer answers than we wanted. The small number of developers participating in our survey also limited our ability to infer from the responses. We note that those responses still greatly agreed on most issues.

Simplifying productivity down to two measures of code growth and effort is sure to miss some dimensions of productivity, *e.g.,* code churn. On the other hand, they do capture more information than simple growth, and are a natural first order approximation measure of energy expended.

We used source code files in each project to identify groups. There are more file types in each project, such as xml files, some of which we may actually be subject to group collaboration, but we had to exclude them to remove bias introduced by files that are not handled by developers, such as “pom.xml”. The number of these files is, however, limited and statistically much smaller than the tens of thousands of source code files in the projects.

We acknowledge the threat to *external validity* of this research, as our findings are based solely on ASF projects and may be affected by cultural and

environmental parameters that may limit the applicability of our findings to other OSS ecosystems such as GitHub.

7 Conclusion

In this paper, we presented an algorithm for tracing group collaboration in OSS. It allowed us to identify hundres of potential team collaborations from Apache data and then to study the associations of teams with software development metrics. We were able to validate a large fraction of the identified teams, demonstrating the practical relevance of the algorithm.

Using that algorithm, we addresed a limited set of topics, including how teamwork associates with measures of productivity. We were able to get significant results demonstrating the practical utility of the algorithm in understanding team development in OSS. Perhaps the most important conclusion coming out of this work is that teams larger than 2 members do form and that we can detect them simply from looking at their commits over time. This technology can thus enable studies on team formation, effectiveness of teams, recruiting and training, as well as early identification of tasks that need team effort. Such studies can lead to results which are more or less expected, but also to some that may not be, *e.g.,* we found here that effort expended is almost universally lowered while developers work together, while the same agreement does not exist for file code growth.

There are several immediate directions for followup research to this work. Our algorithm is still in a state of development and can arguably be improved along several dimensions to make it more useful in practice, *e.g.,* fine-tuning the parameters using data from known team work traces. Adding functionality to this algorithm so it considers the actual code being written by team members, would open additional interesting research avenues into how team members parcel out sub-tasks and how they integrate them into a whole. Finally, studying the trade-off between the benefits of collaboration and its costs and identifying ranges of group sizes lying in the sweet spot could be useful both from the project and individual perspectives.

8 Acknowledgments

The authors would like to thank the members of our DECAL research group and Prof. Qi Xuan for the valuable discussion about the ideas and technical details presented in this paper. We thank also Dr. Bogdan Vasilescu for his contributions in designing the survey and for his insightful comments and feedback on this work, and Mehrdad Afshari for his help in improving the paper. The comments by the anonymous reviewers helped us make this paper better, for which we are thankful. Both authors gratefully acknowledge support from the Air Force Office of Scientific Research, award FA955-11-1-0246.

References

1. P. J. Adams, A. Capiluppi, and C. Boldyreff. Coordination and productivity issues in free software: The role of Brooks' law. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 319–328. IEEE, 2009.
2. B. Al-Ani and H. K. Edwards. A comparative empirical study of communication in distributed and collocated development teams. In *Global Software Engineering, 2008. ICGSE 2008. IEEE International Conference on*, pages 35–44. IEEE, 2008.
3. A. Avritzer and D. J. Paulish. A comparison of commonly used processes for multi-site software development. In *Collaborative Software Engineering*, pages 285–302. Springer, 2010.
4. Y. Baruch. Response rate in academic studies-a comparative analysis. *Human relations*, 52(4):421–438, 1999.
5. C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories*, pages 137–143. ACM, 2006.
6. C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu. Don't touch my code!: examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 4–14. ACM, 2011.
7. C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 24–35. ACM, 2008.
8. N. Blüthgen, F. Menzel, and N. Blüthgen. Measuring specialization in species interaction networks. *BMC ecology*, 6(1):9, 2006.
9. F. P. Brooks, Jr. *The Mythical Man-month (Anniversary Ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
10. B. Caglayan, A. B. Bener, and A. Miranskyy. Emergence of developer teams in the collaboration network. In *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on*, pages 33–40. IEEE, 2013.
11. E. Carmel. *Global software teams: collaborating across borders and time zones*. Prentice Hall PTR, 1999.
12. M. Cataldo and J. D. Herbsleb. Coordination breakdowns and their impact on development productivity and software failures. *Software Engineering, IEEE Transactions on*, 39(3):343–360, 2013.
13. J. Child. Organizational structure, environment and performance: the role of strategic choice. *Sociology*, 6(1):1–22, 1972.
14. P. R. Cohen and H. J. Levesque. *Teamwork*. SRI International Menlo Park, 1991.
15. K. Crowston, Q. Li, K. Wei, U. Y. Eseryel, and J. Howison. Self-organization of teams for free/libre open source software development. *Information and software technology*, 49(6):564–575, 2007.
16. 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*, pages 1277–1286. ACM, 2012.
17. D. Damian, L. Izquierdo, J. Singer, and I. Kwan. Awareness in the wild: Why communication breakdowns occur. In *Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on*, pages 81–90. IEEE, 2007.
18. M. Di Penta, M. Harman, G. Antoniol, and F. Qureshi. The effect of communication overhead on software maintenance project staffing: a search-based approach. In *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, pages 315–324. IEEE, 2007.
19. L. A. Dugatkin. *Cooperation among animals*. Oxford Series in Ecology and Evolution, 1997.
20. M. Foucault, J.-R. Falleri, and X. Blanc. Code ownership in open-source software. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, page 39. ACM, 2014.

21. M. Gharehyazie, D. Posnett, and V. Filkov. Social activities rival patch submission for prediction of developer initiation in oss projects. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 340–349. IEEE, 2013.
22. M. Gharehyazie, D. Posnett, B. Vasilescu, and V. Filkov. Developer initiation and social interactions in oss: A case study of the apache software foundation. *Empirical Software Engineering*, pages 1–36, 2014.
23. M. Goeminne, M. Claes, and T. Mens. A historical dataset for the gnome ecosystem. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 225–228. IEEE Press, 2013.
24. M. Grechanik, J. A. Jones, A. Orso, and A. van der Hoek. Bridging gaps between developers and testers in globally-distributed software development. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 149–154. ACM, 2010.
25. C. Gutwin, R. Penner, and K. Schneider. Group awareness in distributed software development. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 72–81. ACM, 2004.
26. A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. v. Deursen. Communication in open source software development mailing lists. In *MSR*, pages 277–286. IEEE, 2013.
27. J. D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *2007 Future of Software Engineering*, pages 188–198. IEEE Computer Society, 2007.
28. J. D. Herbsleb and R. E. Grinter. Architectures, coordination, and distance: Conway's law and beyond. *IEEE software*, 16(5):63–70, 1999.
29. J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter. An empirical study of global software development: distance and speed. In *Proceedings of the 23rd international conference on software engineering*, pages 81–90. IEEE Computer Society, 2001.
30. J. D. Herbsleb and D. Moitra. Global software development. *Software, IEEE*, 18(2):16–20, 2001.
31. G. Hertel, S. Niedner, and S. Herrmann. Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel. *Research policy*, 32(7):1159–1177, 2003.
32. H. Holmstrom, E. Ó. Conchúir, P. J. Ågerfalk, and B. Fitzgerald. Global software development challenges: A case study on temporal, geographical and socio-cultural distance. In *Global Software Engineering, 2006. ICGSE'06. International Conference on*, pages 3–11. IEEE, 2006.
33. A. Jermakovics, A. Sillitti, and G. Succi. Mining and visualizing developer networks from version control systems. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 24–31. ACM, 2011.
34. T. Kakimoto, Y. Kamei, M. Ohira, and K. Matsumoto. Social network analysis on communications for knowledge collaboration in oss communities. In *Proceedings of the International Workshop on Supporting Knowledge Collaboration in Software Development (KCSD06)*, pages 35–41. Citeseer, 2006.
35. P. Kampstra et al. Beanplot: A boxplot alternative for visual comparison of distributions. *Journal of Statistical Software*, 28(1):1–9, 2008.
36. J. R. Katzenbach. *The wisdom of teams: Creating the high-performance organization*. Harvard Business Press, 1993.
37. B. S. Kuipers and M. C. De Witte. Teamwork: a case study on development and performance. *The International Journal of Human Resource Management*, 16(2):185–201, 2005.
38. F. Lanubile, C. Ebert, R. Prikladnicki, and A. Vizcaíno. Collaboration tools for global software engineering. *IEEE software*, (2):52–55, 2010.
39. K. Luther, K. Caine, K. Ziegler, and A. Bruckman. Why it works (when it works): Success factors in online creative collaboration. In *Proceedings of the 16th ACM international conference on Supporting group work*, pages 1–10. ACM, 2010.
40. W. Maalej and H.-J. Happel. From work to word: How do software developers describe their work? In *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*, pages 121–130. IEEE, 2009.

41. W. Maalej and H.-J. Happel. Can development work describe itself? In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 191–200. IEEE, 2010.
42. I. Mistrík, J. Grundy, A. Van der Hoek, and J. Whitehead. Collaborative software engineering: challenges and prospects. In *Collaborative Software Engineering*, pages 389–403. Springer, 2010.
43. A. Mockus. Succession: Measuring transfer of code and developer productivity. In *Proceedings of the 31st International Conference on Software Engineering*, pages 67–77. IEEE Computer Society, 2009.
44. A. Mockus. Organizational volatility and its effects on software defects. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 117–126. ACM, 2010.
45. N. B. Moe, T. Dingsøyr, and T. Dybå. A teamwork model for understanding an agile team: A case study of a scrum project. *Information and Software Technology*, 52(5):480–491, 2010.
46. N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality: an empirical case study. In *Proceedings of the 30th international conference on Software engineering*, pages 521–530. ACM, 2008.
47. K. Nakakoji, K. Yamada, and E. Giaccardi. Understanding the nature of collaboration in open-source software development. In *Software Engineering Conference, 2005. APSEC'05. 12th Asia-Pacific*, pages 8–pp. IEEE, 2005.
48. K. Nakakoji, Y. Ye, and Y. Yamamoto. Supporting expertise communication in developer-centered collaborative software development environments. In *Collaborative Software Engineering*, pages 219–236. Springer, 2010.
49. T. Nguyen, T. Wolf, and D. Damian. Global software development and delay: Does distance still matter? In *Global Software Engineering, 2008. ICGSE 2008. IEEE International Conference on*, pages 45–54. IEEE, 2008.
50. N. Nohria and R. Eccles. *Networks and organizations: structure, form, and action.* Harvard Business School Press, 1994.
51. D. Pagano and W. Maalej. How do open source communities blog? *Empirical Software Engineering*, 18(6):1090–1124, 2013.
52. S. Panichella, G. Canfora, M. Di Penta, and R. Oliveto. How the evolution of emerging collaborations relates to code changes: An empirical study. In *22nd International Conference on Program Comprehension (ICPC)*. IEEE, 2014.
53. M. Pinzger and H. C. Gall. Dynamic analysis of communication and collaboration in oss projects. In *Collaborative Software Engineering*, pages 265–284. Springer, 2010.
54. D. Posnett, R. D'Souza, P. Devanbu, and V. Filkov. Dual ecological measures of focus in software development. In *35th International Conference on Software Engineering (ICSE)*, pages 452–461. IEEE, 2013.
55. F. Rahman and P. Devanbu. Ownership, experience and defects: a fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 491–500. ACM, 2011.
56. D. Redmiles, A. Van Der Hoek, B. Al-Ani, T. Hildenbrand, S. Quirk, A. Sarma, R. Filho, C. de Souza, and E. Trainer. Continuous coordination-a new paradigm to support globally distributed software development projects. *Wirtschafts Informatik*, 49(1):28, 2007.
57. J. Robertsa, I.-H. Hann, and S. Slaughter. Communication networks in an open source software project. In *Open Source Systems*, pages 297–306. Springer, 2006.
58. E. E. Salas and S. M. Fiore. *Team cognition: Understanding the factors that drive process and performance.* American Psychological Association, 2004.
59. A. Sarma, B. Al-Ani, E. Trainer, R. S. Silva Filho, I. A. da Silva, D. Redmiles, and A. van der Hoek. Continuous coordination tools and their evaluation. In *Collaborative Software Engineering*, pages 153–178. Springer, 2010.
60. A. Sarma, J. Herbsleb, and A. Van Der Hoek. Challenges in measuring, understanding, and achieving social-technical congruence. In *Proceedings of Socio-Technical Congruence Workshop, In Conjuction With the International Conference on Software Engineering*, 2008.

61. W. Scacchi. Collaboration practices and affordances in free/open source software development. In *Collaborative software engineering*, pages 307–327. Springer, 2010.
62. A. Serebrenik and M. van den Brand. Theil index for aggregation of software metrics values. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–9. IEEE, 2010.
63. Y. Takhteyev and A. Hilts. Investigating the geography of open source software through github, 2010.
64. B. Vasilescu, A. Serebrenik, and M. van den Brand. You can't control the unfamiliar: A study on the relations between aggregation techniques for software metrics. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 313–322. IEEE, 2011.
65. J. Whitehead, I. Mistrík, J. Grundy, and A. van der Hoek. Collaborative software engineering: concepts and techniques. In *Collaborative Software Engineering*, pages 1–30. Springer, 2010.
66. E. O. Wilson. What is sociobiology? *Society*, 15(6):10–14, 1978.
67. Q. Xuan, P. T. Devanbu, and V. Filkov. Converging work-talk patterns in online task-oriented communities. *arXiv preprint arXiv:1404.5708*, 2014.
68. Q. Xuan, H. Fang, C. Fu, and V. Filkov. Temporal motifs reveal collaboration patterns in online task-oriented networks. *Physical Review E*, 91(5):052813, 2015.
69. Q. Xuan and V. Filkov. Synchrony in social groups and its benefits. In *Handbook of Human Computation*, pages 791–802. Springer, 2013.
70. Q. Xuan and V. Filkov. Building it together: Synchronous development in OSS. In *Proceedings of the 34th International Conference on Software Engineering*. ACM, 2014.
71. Q. Xuan, M. Gharehyazie, P. T. Devanbu, and V. Filkov. Measuring the effect of social communications on individual working rhythms: A case study of open source software. In *Social Informatics (SocialInformatics), 2012 International Conference on*, pages 78–85. IEEE, 2012.
72. Q. Xuan, A. Okano, P. Devanbu, and V. Filkov. Focus-shifting patterns of oss developers and their congruence with call graphs. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 401–412. ACM, 2014.

Appendix A. Developer Questionnaire

The questionnaire is sent to each individual through email. Each email starts with a proper introduction of the authors, and our research. afterwards, they are asked to complete the form and submit it to us.

ASF Collaborative Development Questionnaire

* Required

How would you describe your involvement in this project?
e.g., project founder, core developer, ...

How frequently did/do you work on this mentioned project? *

- ○ Daily
- ○ Once per 2-3 days
- ○ Once per week
- ○ Less than once per week

What are some typical tasks you carried out in this project? Please give a few examples.
e.g., fixing bugs, implementing a new feature, ...

How do you choose which tasks to work on? Do you choose your own tasks? How do you prioritize which tasks to work on first?

How long did tasks you worked on typically take, from start to finish? *
If you were part of a bigger task, please answer with the overall task in mind

- ○ 1-2 days
- ○ 3-5 days
- ○ A week
- ○ 2 weeks
- ○ Other:

When does work by others influence you / your work directly? *

When it is in the same files you are touching at the time; the same packages; the whole project or something else

- ○ The file(s) I am working on
- ○ The package(s) I am working on
- ○ The whole project
- ○ Other:

Which of your tasks do you consider to be more collaborative than the others?
e.g., bug fixes, adding new features, ...

How many people do collaborative tasks typically involve?

- ○ 2
- ○ 3
- ○ 4
- ○ 5
- ○ 6
- ○ more

How do you coordinate your work with collaborators on the same task? What communication channels do you use?
Do you discuss with them prior to task assignment, during task work, or after task completion?

How do you adjust your working style when collaborating as opposed to during solitary work, if at all?
e.g.,, by committing less frequently, or by pushing smaller commits more frequently, ...

When is it beneficial and when is it detrimental to collaborate with others on the same task?

Please tell us how much you agree or disagree with the following sentences *

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Working on the project was a collaborative effort	○	○	○	○	○
You actively attempted to "team up" with others to complete tasks	○	○	○	○	○
Collaboration increases productivity	○	○	○	○	○
Collaboration increases merge conflicts and introduces some difficulties	○	○	○	○	○
Collaboration requires extra coordination and communication	○	○	○	○	○

Appendix B. Verification of Data Mining Scripts

Our scripts are based on scripts developed by Bird *et al.*, which we have slighlty modified to fit our purposes. Both ours and their scripts are available at `http://www.gharehyazie.com/supplementary/teamwork/miningscripts/`. As this data gathering step is critical to the analyses downstream, we proceeded to verify their accuracy. To that end, we randomly selected three months (June 2008, April 2009, and Feburary 2010) and three of our 26 projects (Abdera, Harmony and Cayenne). We then manually iterated over all of the messages by those selected projects during those selected time periods. Overall about 1200 messages were inspected during this process, as follows.

We observed the message senders, subject, timestamp, thread IDs, and body. This information was then compared to the corresponding entries for the messages in the projects' mailing list archive available at `http://mail-archives.apache.org/mod_mbox/`. While almost everything was consistent the original archive, two issues were discovered:

1. The timestamp of messages stored in our database were off by a few hours compared to the archives. Upon further investigation, we identified the issue to be the way we parse the timezone information. This inconsistency does not affect our results since it results in a time discrepancy in message timestamps of at most one day and our study is insensitive to this resolution of time.
2. The last message of each month was not recorded in our database. This resulted in a difference of 12 messages per project per year between our database and the actual archives, a difference of 1%.