Building It Together: Synchronous Development in OSS

Qi Xuan*† qxuan@ucdavis.edu

Vladimir Filkov† filkov@cs.ucdavis.edu

*Department of Automation
Zhejiang University of Technology
Hangzhou 310023, China

†Department of Computer Science
University of California, Davis
Davis, CA 95616-8562, USA

ABSTRACT

In distributed software development synchronized actions are important for completion of complex, interleaved tasks that require the abilities of multiple people. Synchronous development is manifested when file commits by two developers are close together in time and modify the same files. Here we propose quantitative methods for identifying synchronized activities in OSS projects, and use them to relate developer synchronization with effective productivity and communication. In particular, we define co-commit bursts and communication bursts, as intervals of time rich in co-commit and correspondence activities, respectively, and construct from them smoothed time series which can be, subsequently, correlated to discover synchrony. We find that synchronized co-commits between developers are associated with their effective productivity and coordination: during co-commit bursts, vs. at other times, the project size grows faster even though the overall coding effort slows down. We also find strong correlation between synchronized co-commits and communication, that is, for pairs of developers, more co-commit bursts are accompanied with more communication bursts, and their relationship follows closely a linear model. In addition, synchronized co-commits and communication activities occur very close together in time, thus, they can also be thought of as synchronizing each other. This study can help with better understanding collaborative mechanisms in OSS and the role communication plays in distributed software engineering.

Categories and Subject Descriptors

D.2.9 [Software Engineering]: Management—Programming teams; D.2.8 [Software Engineering]: Metrics—Process metrics

General Terms

Theory, Measurement, Management

Keywords

OSS, collaboration, communication, synchronization

1. INTRODUCTION

Software engineering projects depend on collaboration [19, 24, 30, 31, 45, 51], as almost any non-trivial software requires the effort of multiple developers to design it, implement features, maintain revisions, and so on. Open Source Software (OSS) projects are examples of organized software development where collaboration is essential. Collaboration is well defined and organized in traditional companies [12, 13, 52], however, its mechanism in OSS projects is still unclear to date, although it is considered to play a key role in the success of many OSS projects [7, 43], such as Apache, Mozilla, Eclipse, etc. How does collaboration come to pass in OSS, and why? While lacking formal organization, these projects do exhibit a social network among the participants enabling them to accomplish complex tasks [7]. For example, often multiple developers work on the same or related files around the same time, and these activities, or collaborations are accompanied by task-coordination via remote communications, through different electronic tools, such as emails, instant messaging, and web-based applications. So, collaboration and communication in OSS should be strongly connected, but how can we quantify that?

Many OSS projects have existed for a long time, e.g., more than ten years, and are comprised of large numbers of files, while their organizations are highly dynamic [6, 16, 34, 41], i.e., developers join and quit frequently. Thus, to assert that there is a collaborative relationship between two developers we need more than just evidence that they simply contributed to the same project or same files; we need to know whether their activities occurred close together in time. Note that here we only focus on file-level collaboration [46] in order to observe the generative process of a project more carefully. A collaboration in OSS projects, therefore, involves, at the minimum, a pair of developers, their possible contribution to the same artifacts, and overlapping times of contribution. To emphasize the synchrony of such a collaborative activity and distinguish it from the traditional definition of collaboration, here we call it synchronous development. Fortunately, most activities of developers, including code commits and communication, are all recorded in OSS repositories [4, 5, 27, 50]. These valuable data sets provide an excellent opportunity for us to define synchronous development between developers in an objective way, to study the relationship between synchronous development and communication, and further reveal their effects on productivity in these projects.

Here, we study synchronous development in OSS projects, its effect on a measure of code size, the number of gener-
ated lines of code (LOC) [28] by developers, and the association between developer communication and their synchronous development. Specifically, we make a dual contribution. First, we present a theoretical framework to quantitatively study synchronous development and communication, based on intervals of time rich in co-commits and correspondence activities between pairs of developers, which we call, respectively, co-commit bursts, or C-bursts, and email communication bursts, or E-bursts. Time-series smoothing and correlation allow us to identify synchronous patterns across bursts.

Our second, empirical contribution, is in applying the quantitative framework to data from six OSS projects, with the following results:

- We find that during a synchronous development more LOC are generated and fewer are deleted, i.e., the projects grow faster, less coding effort is expended, and the coordination is more effective;
- We show that synchronous development and communication are positively correlated and accelerate each other;
- We present a metric to compare communication tools as effectors on task-coordination between developers;
- We provide case studies by analyzing the content of email messages in support of the quantitative results.

The rest of the paper is organized as follows. Next, we discuss our research questions. In Section 3, we present our methods, followed by the results in Section 4. Then, we discuss the possible threats to the validity in Section 5, followed by a section on related work in Section 6. Finally, the paper is concluded in Section 7.

2. RESEARCH QUESTIONS

The C-burst and E-burst methodology, which we present in the next section, is useful for quantifying emerging synchronous relationships between actions of participants in complex distributed systems, like OSS projects. We put it to use in answering questions related to synchronization of developers in OSS projects.

To make the case for synchronous development we first ask if there is sufficient evidence in the data to support the assertion that synchronous development is more than random commits by pairs of developers.

Research Question 1: Is synchronous development more than a random phenomenon? That is, does it occur significantly more frequently in real OSS projects than it would by chance?

Next, we consider the effects that synchronous development might have on efforts expended and on code growth in OSS projects. In an OSS project, it is plausible that developers can contribute more when they participate in synchronous development. Here, to quantify this phenomenon, we follow previous studies and use the lines of code, or LOC, to measure the effort developers expend as well as the project size and its growth. In a simplified, but easy to quantify, sense, each commit activity of a developer consists of two parts: adding new code and deleting old code. If we let \( L_{\text{Add}} \) and \( L_{\text{Delete}} \) denote the LOC added and deleted per commit, respectively, then the change in LOC, or project size, for each commit is:

\[
\Delta L = L_{\text{Add}} - L_{\text{Delete}}.
\]

Similarly, we can also define the effort expended as:

\[
\Delta W = L_{\text{Add}} + L_{\text{Delete}}.
\]

We note that effort may go into appropriately deleting lines of code, which may result in a small, if any, size difference to the project code. Then, we ask,

Research Question 2: Do projects grow more, in terms of larger \( \Delta L \), per commit, when developers participate in synchronous development? Do developers expend more or less effort, in terms of \( \Delta W \), during synchronous development?

Software projects are inherently cooperative, which requires many software engineers to coordinate their efforts in order to produce a large software system [51]. Collaboration between developers always involves coordination challenges, which have been studied extensively [22, 25, 30], including careful interview studies [23]. In OSS, coordination means respecting others’ work while making one’s own work more compatible with theirs. In other words, a developer adding his lines of code tries not to affect much others’ lines. This, we expect, should be easier done during co-commit bursts.

Research Question 3: Are developers more effective in coordination, in terms of smaller \( L_{\text{Delete}} \) per commit, during co-commit bursts than outside of co-commit bursts?

Communication is vital in making collaboration effective. Since software developers in OSS projects work from wherever they happen to be, electronic communications, such as email, rather than traditional face-to-face conversations, are preferred [47]. The public availability of email traces makes it possible to analyze the relationship between commit and communication activities. We have already found that communication and commits accelerate each other in Apache OSS projects [54]. Since communications are necessary for collaboration, here it is natural to assume that synchronous developments accompany communications very closely.

Research Question 4: What is the relationship between C-bursts and E-bursts? Are more co-commit bursts always accompanied with more email bursts?

Further, if there exists a positive relationship between the C- and E-bursts, it is plausible that communication and synchronous development are positively coupled, i.e., having more of one increases the other.

Research Question 5: Are communication and synchronous development activities positively coupled (in a dynamic sense)? That is, will increase in one result in increase in the other, shortly thereafter?
3. METHODOLOGY

Here we describe the mathematical framework we use to make the concept of synchronous development more practical, based on time-series of commit and communication activities of developers.

For an OSS project, we denote by $D$ the set of developers and by $F$ the set of files. For each developer $d \in D$, let $T(d)$ be the sequence of their commit times. Since a developer may submit changes to several files at the same time, let $f_i(d) \subseteq F$ the subset of files to which developer $d$ made changes at the $i$'th commit. For a pair of developers $d_a$ and $d_b$, we denote by $\Omega(d_a, d_b)$ the sequence of their interleaved email communications’ times. We consider all communications undirected, thus, $\Omega(d_a, d_b) \equiv \Omega(d_b, d_a)$, for any $d_a, d_b$.

We denote by $\xi$ a fixed time-window, based on which we define the time-series of events we call co-commit bursts, or C-bursts, and email bursts, or E-bursts.

Co-commit Bursts A co-commit burst, or C-burst, is a sequence of temporally interleaved commit activities of two developers such that each commit of the first developer is followed closely, within time $\xi$, by a commit of the second developer, and of the files the two commits modify at least one is the same in both. More precisely, in a C-burst sequence, for every commit activity of the first developer, $d_a$, modifying a set of files $f_i(d_a)$ at time $t_i(d_a)$ in the burst, there is at least one commit activity from the other developer $d_b$ on the file set $f_j(d_b)$ at time $t_j(d_b)$ in the burst, and no such activities outside of the burst exist satisfying $|t_i(d_a) - t_j(d_b)| \leq \xi$ and $f_i(d_a) \cap f_j(d_b) \neq \emptyset$.

Based on the above definition we use a one-dimensional clustering algorithm to identify C-bursts, as follows:

1. For each pair of developers $d_a$ and $d_b$, $a \neq b$, we denote by $U$ the sub-burst set, and set $U = \emptyset$.

2. For each commit of $d_a$ at time $t_i(d_a) \in T(d_a)$, it is in a sub-burst if there is at least one commit of $d_b$ at time $t_j(d_b) \in T(d_b)$, such that $|t_i(d_a) - t_j(d_b)| \leq \xi$, and the same files were changed, i.e., $f_i(d_a) \cap f_j(d_b) \neq \emptyset$. We group all such commits of $d_a$, as well as the commit of $d_b$, into the sub-burst, and record its time interval from the first commit to the last in the sub-burst. Then, we add this sub-burst to $U$.

3. For all sub-bursts in $U$, we merge those with overlapping time intervals, and update the corresponding start and end times of the burst.

4. The final set of C-bursts for $d_a$ and $d_b$ is denoted by $B^C(d_a,d_b)$, with the occurrence time of each burst given by the mean of its start and end times.

We note that the simpler approach of dividing developers’ overlapping active-time intervals into successive time-windows of the same length, and then checking for commits to the same files in the same time-windows does not work in general, as it results in breaking up a synchronous development, especially when the synchronous development lasts for a longer time.

Email Bursts An email burst, or E-burst, is a sequence of temporally interleaved email communication activities of two developers, such that the time interval between any two successive emails in the burst is smaller than some minimal time, say $\xi$. Thus, for every pair of developers $d_a$ and $d_b$, all their E-bursts are just the subsequences remaining after exciting all intervals longer than $\xi$ from their sequence of email communication activities occurring at $\Omega(d_a, d_b)$.

Note that an E-burst is different from an email thread [43] which is the group of emails referring to the same subject. In contrast, an E-burst is determined by the time that emails were sent and not their content.

3.1 Comparing Bursts via Smoothed Curves

Work related communications may occur before or after the commit activities being discussed, as, e.g., plans or conclusions, respectively. Hence, C-bursts and E-bursts can occur at non-overlapping time periods which makes it difficult to gainfully correlate those time-series. In addition, the bursts are, in general, non-regular time series, and also discrete, and noisy.

To extract trends and patterns from such challenging time-series, and at the same time reduce the effect of noise we use smoothing. This technique reduces complexity in time-series data and allows for easy comparison of the smoothed curves. For the C-bursts and E-bursts we use Gaussian smoothing, based on a symmetric bell curve [38], defined as:

$$f(x, \zeta) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(x-\zeta)^2}{2\sigma^2}},$$

(3)

where $\zeta$ is the position of the peak and $\sigma$ controls the width of the curve. An important character of the Gaussian function is that it quickly decreases to zero when $x$ goes outside of the range $[\zeta - \sigma, \zeta + \sigma]$.

For every two developers, let $\Gamma^C$ and $\Gamma^E$ be the sets of occurring times of their co-commit and email bursts, respectively. Then, the corresponding smoothed bursts, C-curve and E-curve, are defined as (|·| is set cardinality)

$$\varphi^C(x) = \frac{1}{|\Gamma^C|} \sum_{\zeta \in \Gamma^C} f(x, \zeta), \varphi^E(x) = \frac{1}{|\Gamma^E|} \sum_{\zeta \in \Gamma^E} f(x, \zeta).$$

(4)

If we let $\Delta$ be the time interval between the minimum time and maximum time in $\Gamma^C \cup \Gamma^E$, then the corresponding centralized curves on the whole $\Delta$ interval are calculated by

$$\phi^C(x) = \varphi^C(x) - \frac{1}{\Delta} \int_{L}^{U} \varphi^C(x) dx,$n

(5)

$$\phi^E(x) = \varphi^E(x) - \frac{1}{\Delta} \int_{L}^{U} \varphi^E(x) dx.$n

(6)

Then, we can measure the synchronization between the two curves by calculating the Pearson correlation coefficient [9], defined as

$$R = \frac{\int_{L}^{U} \phi^C(x) \phi^E(x) dx}{\sqrt{\int_{L}^{U} [\phi^C(x)]^2 dx} \sqrt{\int_{L}^{U} [\phi^E(x)]^2 dx}}.$n

(7)

Based on this definition, $R$ is 1 when $\phi^C(x) \equiv \phi^E(x)$; it is -1 when $\phi^C(x) \equiv -\phi^E(x)$, and it is 0 when the two curves are independent of each other.

3.2 Null Models

The values of the Pearson correlation coefficients as calculated by Eq. (7) tell us the magnitude of the synchronization between synchronous development and communication activities of pairwise developers, but they don’t specify if the

1We expect that the results will not change much if other kernel functions are adopted.
synchronization is significant statistically. To calculate the significance we conduct simulations to generate randomized instances of our data.

We simulate random email communication time-series by randomizing the time intervals between successive activities, using a method we recently developed [54], summarized here for completeness: First, for each pair of developers, let their communications occur at the successive time $t_1, t_2, \ldots, t_n$ in reality. Denote the $n-1$ ordered inter-communication time intervals by $\Delta t_{ik} = t_{i+1} - t_i$, with $k = 1, 2, \ldots, n-1$.

Then, randomly rearrange them to get a new sequence of time intervals, denoted by $\Delta t^*_k, k = 1, 2, \ldots, n-1$. Finally, weld these new ordered time intervals together to get a new time-series $t^*_1, t^*_2, \ldots, t^*_n$, satisfying

$$t^*_k = t_k, \quad t^*_k = t^*_{k-1} + \Delta t^*_k, \quad k \geq 2. \quad (8)$$

Such simulated email communication time-series have the same distribution of time intervals as the empirical one. We generate simulated E-curves with the above, and then calculate the correlation coefficients between the real C-curves and the corresponding simulated E-curves for comparison.

Note that, to make the argument that synchronous developments occurs more frequently than by chance, we also randomize the commit activities for each developer by the same method, while keeping the order of each activity and the associated files they access. Then, we enumerate the simulated C-bursts for all pairs of developers and compare them with the numbers of real C-bursts. In both cases, the communication or committing activities are randomized one hundred times.

### 4. RESULTS AND DISCUSSION

We obtained data for 31 OSS projects from the Apache Software Foundation on March 24th, 2012. For each project, the commit activities of developers on different files are gathered from the corresponding Git repository while the email communication activities are gathered from the online developer mailing lists. For each commit activity, we recorded the developer ID, file ID, file type, the exact submitting time in seconds, and the numbers of added and deleted LOC in each file. For each communication activity, we recorded the sender ID, receiver ID, and the sending time in seconds. Note that, developers may have multiple aliases, which were resolved by using a semi-automatic approach [5]. In this paper, we focus on the six projects with most developers: Ant, Axis2 Java, Cxf, Derby, Lucene, and Openejb, in order to get most meaningful statistical results. Several of their basic properties are presented in Table 1.

#### Table 1: Basic properties of the six OSS projects.

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
<th>Time period</th>
<th>#Developers</th>
<th>#Files</th>
<th>#Commits</th>
<th>#Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant</td>
<td>Software tool</td>
<td>2000/01/13-2012/03/16</td>
<td>44</td>
<td>11620</td>
<td>14697</td>
<td>9398</td>
</tr>
<tr>
<td>Axis2 Java</td>
<td>Web services engine</td>
<td>2001/10/04-2012/03/18</td>
<td>72</td>
<td>129978</td>
<td>13466</td>
<td>11865</td>
</tr>
<tr>
<td>Cxf</td>
<td>Web services framework</td>
<td>2005/07/22-2012/03/16</td>
<td>45</td>
<td>37867</td>
<td>14288</td>
<td>4625</td>
</tr>
<tr>
<td>Derby</td>
<td>Database management system</td>
<td>2004/08/11-2012/03/22</td>
<td>35</td>
<td>6563</td>
<td>8301</td>
<td>25057</td>
</tr>
<tr>
<td>Lucene</td>
<td>Search software</td>
<td>2001/09/11-2012/03/23</td>
<td>41</td>
<td>6674</td>
<td>5337</td>
<td>17708</td>
</tr>
<tr>
<td>Openejb</td>
<td>Container system and server</td>
<td>2002/01/18-2012/03/22</td>
<td>38</td>
<td>43960</td>
<td>7801</td>
<td>5312</td>
</tr>
</tbody>
</table>

#### Table 2: T-tests of the difference between C-bursts in real and simulated data for developer pairs, with different time-windows.

<table>
<thead>
<tr>
<th>Project</th>
<th>$\xi$ (day)</th>
<th>Real</th>
<th>Simulated</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant</td>
<td>1</td>
<td>4.8868</td>
<td>0.5226</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5.8178</td>
<td>1.1538</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>5.2654</td>
<td>1.3572</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td>Axis2 Java</td>
<td>1</td>
<td>2.4983</td>
<td>0.3529</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2.8045</td>
<td>0.7335</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2.5207</td>
<td>0.8570</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td>Cxf</td>
<td>1</td>
<td>5.1783</td>
<td>0.3028</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5.6818</td>
<td>0.6620</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>4.8298</td>
<td>0.7658</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td>Derby</td>
<td>1</td>
<td>2.3100</td>
<td>0.5715</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4.1624</td>
<td>1.5424</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>4.8635</td>
<td>1.9674</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td>Lucene</td>
<td>1</td>
<td>3.6504</td>
<td>0.5967</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4.3571</td>
<td>0.9328</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>3.8118</td>
<td>0.9279</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td>Openejb</td>
<td>1</td>
<td>2.4800</td>
<td>0.3261</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3.5442</td>
<td>0.7192</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>3.6379</td>
<td>0.8404</td>
<td>&lt;0.00001</td>
</tr>
</tbody>
</table>

### 4.1 Synchronous Development Is Real

Based on the methods described above, we identify the C-bursts for each pair of developers. To show that synchronous development is a significant emergence rather than a random phenomenon that developers committed to the same files at close time just by chance, we created simulated commit time-series for each developer with our null model. Then, by the same method, we also generated C-bursts for each pair of developers based on their simulated co-commit time-series. We find that developers exhibit synchronous development much more frequently than in the simulated case, as indicated by the T-test results presented in Table 2. This answers in the positive Research Question 1. Note that the number of C-bursts between a pair of developers is dependent on the time window, e.g., it goes to zero when $\xi \to 0$, i.e., developers cannot commit to the same files at the exactly same time, and it goes to one when $\xi \to \infty$, if they commit to at least one common file over the whole project history. In reality, there may be several C-bursts for a pair of developers.

There are some practical limitations on the size of the burst time-window. It cannot be too narrow, otherwise the daily work-rhythm and different time zones may have a significant effect on the results. It also cannot be too wide, otherwise the development cannot be considered synchronized. Therefore, we chose $1 \leq \xi \leq 10$ in this paper.
Table 3: The fraction of C-bursts in which at least one .java file was committed to by both developers.

<table>
<thead>
<tr>
<th>Projects</th>
<th>Ratio of .java (%)</th>
<th>Coverage ratio on C-bursts (%) under different time-windows (day)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant</td>
<td>64.6%</td>
<td>47.9% 47.9% 49.3% 50.3% 49.8% 51.1% 51.3% 52.1% 51.9% 51.9% 51.9%</td>
</tr>
<tr>
<td>Axis2_java</td>
<td>60.4%</td>
<td>73.2% 73.7% 74.2% 74.5% 74.5% 74.6% 74.4% 73.9% 73.9% 73.9% 73.9%</td>
</tr>
<tr>
<td>Cxf</td>
<td>55.9%</td>
<td>75.7% 75.1% 75.9% 75.8% 75.8% 76.3% 76.3% 76.7% 76.9% 76.2% 76.2%</td>
</tr>
<tr>
<td>Derby</td>
<td>49.4%</td>
<td>75.1% 76.2% 76.4% 76.5% 77.2% 77.5% 78.1% 79.3% 79.4% 79.7% 79.7%</td>
</tr>
<tr>
<td>Lucene</td>
<td>73.2%</td>
<td>41.4% 42.8% 42.9% 45.0% 43.7% 41.7% 41.7% 42.2% 43.7% 42.9% 42.9%</td>
</tr>
<tr>
<td>Openejb</td>
<td>79.4%</td>
<td>57.5% 56.6% 56.3% 57.5% 56.9% 55.2% 54.2% 53.8% 53.8% 53.6% 53.6%</td>
</tr>
</tbody>
</table>

4.2 Synchronous Development, Code Growth, and Effort

Before we present the main results, it is worthwhile to first investigate the file types where synchronous developments occurred. Generally, there are always dozens of file types in each OSS project, including .java, .xml, .txt, and so on. As expected, most C-bursts involved .java source files, i.e., most pairwise developers committed to at least one .java file when they work synchronously. The ratios of .java files and the coverage ratios of these files on C-bursts (over all C-bursts) for the six OSS projects under different time-windows are presented in Table 3. We can see that, besides synchronous development on source code, developers also spent at least one quarter of their C-burst time on other kinds of files, e.g., writing documentation. Since .java files are dominant in these OSS projects and most of the creations of developers are reflected in their contributed source codes stored in these files, in this part, we will mainly focus on the effect of synchronous development on .java files.

Here, we also do not consider all the .java files, because some of them may be committed to by developers separately all the time and never be committed to by more than one developer in the same C-burst, and thus they cannot provide useful information for our study on identifying synchronous development commit. Therefore, we will just focus on those .java files that were committed to by two or more developers in at least one same C-burst. For an OSS project, we gather a list of files such that each is co-committed to within C-bursts by two or more developers at some times and committed to by a single developer outside of C-bursts at some others. Then, we obtain the synchronous and non-synchronous commit activities for all developers on each of these files, and calculate the average number of added LOC, $L_{Add}$, deleted LOC, $L_{Delete}$, and $\Delta L$ per commit.

Figure 1 shows the box-and-whisker diagrams of $\Delta L$ during and outside of co-commit bursts, under different time-windows from one day to ten days to visualize their differences in the six projects. We also use the T-test to ascertain if the differences are significant. The statistical significance for each case is marked on the x-axis. E.g., "*" means $p < 0.05$, "**" means $p < 0.01$, "***" means $p < 0.001$, and "****" means $p < 0.0001$.

![Figure 1: The box-and-whisker diagrams of the average $\Delta L$ per commit during and outside of co-commit bursts, under different time-windows, on the x-axis, where *="p < 0.05", **="p < 0.01", ***="p < 0.001", ****="p < 0.0001".](image-url)

In general, we can see that for the four of six projects, including Ant, Axis2.java, Cxf, and Openejb, the overall code grows faster during synchronous development, in most cases, answering largely positively Research Question 2. More interestingly, we find that, for those relatively older projects including Ant, Axis2.java, and Openejb, synchronous de-
development has positive $\Delta L$ while non-synchronous development has negative $\Delta L$, in most cases. For the relatively younger projects Cxf and Derby, most commit activities, synchronous or not, have positive $\Delta L$. This interesting finding may indicate that these projects are in different stages of development. The less significant results in Lucene may be partly attributed to the relatively fewer files and the lower coverage ratio of .java files on C-bursts in this project. Considering all types of files together, we can still observe significantly faster code growth during synchronous development in that project when $\xi = 3, 6, 7$.

Similarly, we considered $\Delta W = L_{Add} + L_{Delete}$, as a proxy for the total effort expended by developers. We found that for all projects except Cxf $\Delta W$ was significantly smaller during synchronous development than at other times. Again, we hypothesize that the age of the project might be a significant determinant to these results, but that needs further study. We omit the boxplots due to space constraints.

Figure 2 shows the box-and-whisker diagrams of $L_{Delete}$ during and outside of co-commit bursts, under different time-windows. We can see that, for all six projects, developers delete significantly fewer LOC when they participate in synchronous development in large majority of the cases, answering positively Research Question 3. Note that we also did these experiments by considering all types of files together, and obtained similar results.

### 4.2.1 Case Study

To illustrate what goes on during synchronous development, we looked at the content of emails between a pair of developers in Derby. They have the most E-bursts that are close to their C-bursts in the project, i.e., when the time window is set to be $\xi = 1$ (day), we find 84 E-bursts near and 184 E-bursts far from their C-bursts. An E-burst is considered near a C-burst if their time periods overlap or the gap between them is not larger than some threshold.

Here, the threshold is set to be 10 days, in order to get a comparable number of E-bursts near C-bursts. Although much coordination among developers may not be explicitly addressed in their emails, we still find a number of such examples in the examined emails. We find that the developers provide suggestions\(^3\) to improve the current work in 33/84 E-bursts near C-bursts, while the number significantly decreases to 17/184 for those far from C-bursts. Due to space considerations we just provide the following two examples.

---

### Case #1 in Derby [2010/05/12 2:58 PM]

**Message**: I think this code either should be changed to work regardless of ....... or the assumption that ...... should be documented in the comments.

**Reply**: As a consequence of this patch, ...... it may arise ...... I have not been able to figure out ...... but if it may arise when we re-enable ...... we may want to handle this situation more gracefully. For instance, we may want to add two need kinds of Restriction subclasses to model the boolean literals.

---

### Case #2 in Derby [2010/12/08 6:08 PM]

**Message**: I have found ...... to be a very useful class for ...... I would like to ...... and then expose it as part of Derby’s public api.

**Reply**: That reminds me why we decided not to include ...... I would prefer a solution that didn’t require code changes in the application when moving from one Java version to another.

---

\(^3\)Such suggestions always include the words such as we should probably, we may want to add, we could perhaps have, I would prefer, in addition to ...... we need to, we would then do something like this, would it be better/worthwhile, it would be nice to, here are some options for, etc.
Table 4: Correlation test for the numbers of C-bursts and E-bursts under different time-windows (day). Here, $R_L$ and $R_U$ denote the lower and upper bounds, respectively, for a 95% confidence interval for the correlation coefficient.

<table>
<thead>
<tr>
<th>Project</th>
<th>$\xi$</th>
<th>$R_C$</th>
<th>$R_L$</th>
<th>$R_U$</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant</td>
<td>1</td>
<td>0.7874</td>
<td>0.7613</td>
<td>0.8110</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.7848</td>
<td>0.7584</td>
<td>0.8057</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.7712</td>
<td>0.7441</td>
<td>0.7958</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td>Axis2_java</td>
<td>1</td>
<td>0.6377</td>
<td>0.6345</td>
<td>0.6798</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.7008</td>
<td>0.6802</td>
<td>0.7203</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.6718</td>
<td>0.6497</td>
<td>0.6929</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td>Cxf</td>
<td>1</td>
<td>0.7336</td>
<td>0.7035</td>
<td>0.7611</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.7850</td>
<td>0.7599</td>
<td>0.8078</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.7903</td>
<td>0.7657</td>
<td>0.8126</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td>Derby</td>
<td>1</td>
<td>0.7272</td>
<td>0.6870</td>
<td>0.7530</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.7324</td>
<td>0.6928</td>
<td>0.7676</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.7305</td>
<td>0.6906</td>
<td>0.7659</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td>Lucene</td>
<td>1</td>
<td>0.7523</td>
<td>0.7192</td>
<td>0.7819</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.7752</td>
<td>0.7447</td>
<td>0.8024</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.7404</td>
<td>0.7070</td>
<td>0.7706</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td>Openjb</td>
<td>1</td>
<td>0.5080</td>
<td>0.4458</td>
<td>0.5635</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.6039</td>
<td>0.5518</td>
<td>0.6513</td>
<td>&lt;0.00001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.6065</td>
<td>0.5562</td>
<td>0.6524</td>
<td>&lt;0.00001</td>
</tr>
</tbody>
</table>

Case #1 indicates that developers remind each other to do more related work, and sometimes to add more comments to make their work more clear for both of them. And case #2 indicates that developers indeed prefer solutions that didn’t require more code changes, i.e., they delete less code. Such suggestions are provided more frequently when the developers work synchronously than at other times, in support of the numerical results on Research Question 2 and Research Question 3.

4.3 The Role of Communication

As we established above, synchronous development is more than a random phenomenon, therefore, communication is expected to play an important role in this process. In other words, we expect that more communication is needed to achieve and maintain synchronous development, and that these two activities synchronize each other, i.e., they occur close together in time.

First, by the correlation test for the numbers of C-bursts and E-bursts, we find that these values in all the six projects under different time-windows are significantly correlated with each other with relatively high correlation coefficients (>0.5) and small p-values (<0.00001), as presented in Table 4, answering Research Question 4. In fact, when we consider all pairs of developers in the six projects together, the relationship between the number of C-bursts, denoted by $N^C$, and that of E-bursts, denoted by $N^E$, for pairwise developers can be fitted very well with the following linear model:

$$N^C = \alpha N^E + \beta,$$

with parameters (95% confidence bounds) equal to $\alpha = 0.2727$ (0.2645, 0.2810) and $\beta = 0.1331$ (0.0369, 0.2293) when the time-window is set to be $\xi = 5$ (day), which means that, on average, we would expect one burst of co-commits to the same files for every four bursts of email communications between two developers in these projects. The data and fit are shown in Figure 3, where we also show the goodness of fit, including SSE, R-square, Adjusted R-square, and RMSE. In addition to the data, on the plot we also show the data smoothed by moving averages, with span set to 11. The circled samples are discussed in the case studies.

Because the number of E-bursts is a monotonically decreasing function of the time-window, the parameter $\alpha$ may steadily increase as the time-window increases. For example, if we consider the parameter will be $\alpha = 0.1216$ (0.1168, 0.1263) and $\alpha = 0.3538$ (0.3442, 0.3634) when the time-window is set to be $\xi = 1$ (day) and $\xi = 10$ (day), respectively. Since we only consider the strictest synchronous development that developers must co-commit to the exactly same files at close times, it is not surprising to observe that the slope of the linear model is much smaller than 1 even for a large time-window. This is because, in reality, communications between developers may be about co-committing synchronously on functionally related [24], rather than exactly the same files, which we do not consider here.

4.3.1 Case Study

Although the numbers of C-bursts and E-bursts are linearly correlated on average, we can still find pairs of developers that have extremely high ratios of C-bursts versus E-bursts, and vice-versa, which may be determined by their distinct roles in these projects, as indicated in Figure 3.

We selected five pairs of developers with the highest C-to E-burst ratios under the condition that the number of E-bursts is larger than 50, and find that the associated seven developers are distributed in two projects, Ant and Cxf, and centered by two Apache PMC chairs who spent quite a bit of time on committing to files in these projects. We present two of their emails in the following to show that they seem engaged in commit activities to an extraordinary degree.
are indeed significantly larger than those in the simulated
we can see that the correlation coefficients in the real case
our null model. The results are presented in Table 5, where
the six projects, and then compare them with the results ob-
numbers of C-bursts and E-bursts both larger than five for
from each other.

tends to 0 if these two kinds of activities are independent.

correlation coefficient tends to 0 if these two kinds of activities are independent from each other. In
this experiment, we set the time-window $\xi = 1$ (day) and the curve parameter $\sigma = 10$, and the results are stable to
changes to other appropriate values. Note that correlation
coefficient tends to 0 when $\sigma \to 0$ and tends to 1 when $\sigma \to \infty$, therefore $\sigma$ cannot be chosen too small or too large.

4.4 Case Study

The synchronization between email communication and
synchronous development is also partly reflected in the con-
ents of some C-bursts and the accompanied E-bursts be-
tween developers. We manually sample two such cases in
Lucene as examples to see whether the developers discussed
the same files that they both committed to in the C-burst.
The time-window is set to be $\xi = 1$ (day). The first C-burst
occurred on Oct 30, 2009, from 12:45 PM to 5:11 PM with the
accompanied E-burst from 9:36 AM, Oct 29 to 9:37 PM, Oct 30. And the second C-burst occurred from 8:29 AM, Nov 16 to 3:03 PM, Nov 17 with the accompanied E-burst from 4:42 PM, Nov 15 to 9:54, Nov 17.

We find that the two pairs of developers commit to the
same file RussianLowerCaseFilter.java and StandardTokenizerImpl.jflex in the respective C-bursts, while the accompa-
nied E-bursts indeed recorded the discussion of the devel-
opers about these files and the related ones. Part of their
discussions in the E-bursts are listed below.

4.4.1 Case Study

Finally, how closely do synchronous development and com-
munication activities follow each other? Recall, we use C-
curves and E-curves as smoothed versions of the cor-
responding bursts, Eqs. (5) and (6), respectively, and then measure the synchronization between synchronous development and email communication activities by calculating the correlation coefficient between the corresponding C-curves and E-curves, Eq. (7). The value of the correlation coefficient tends to 1 if the numbers of C-bursts and E-bursts are exactly the same and they occur together at very close times, while it
tends to 0 if these two kinds of activities are independent from each other.

We calculate the correlation coefficients of C-curves and the corresponding E-curves for all pairs of developers with numbers of C-bursts and E-bursts both larger than five for
the six projects, and then compare them with the results ob-
tained from the corresponding simulated curves created by
our null model. The results are presented in Table 5, where
we can see that the correlation coefficients in the real case
are indeed significantly larger than those in the simulated

\[ \rho = \frac{\text{cov}(X,Y)}{\text{std}(X) \cdot \text{std}(Y)} \]

\[ \text{cov}(X,Y) = \frac{1}{N-1} \sum_{i=1}^{N} (X_i - \mu_X)(Y_i - \mu_Y) \]

\[ \text{std}(X) = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (X_i - \mu_X)^2} \]

\[ \mu_X = \frac{1}{N} \sum_{i=1}^{N} X_i \]

\[ \mu_Y = \frac{1}{N} \sum_{i=1}^{N} Y_i \]

\[ \xi = \text{time-window} \]

\[ \sigma = \text{curve parameter} \]

\[ \text{Table 5: T-test for the difference between the corre-
lation coefficients of C-curves and the corresponding
E-curves in the real and simulated cases. Here, only
the pairwise developers with both numbers of C-
bursts and E-bursts larger than five are considered.} \]

<table>
<thead>
<tr>
<th>Projects</th>
<th>Real</th>
<th>Simulated</th>
<th>T-value</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant</td>
<td>0.4065</td>
<td>$-8.9 \times 10^{-3}$</td>
<td>14.2</td>
<td>$&lt;0.00001$</td>
</tr>
<tr>
<td>Axis2.java</td>
<td>0.3642</td>
<td>$9.4 \times 10^{-3}$</td>
<td>15.7</td>
<td>$&lt;0.00001$</td>
</tr>
<tr>
<td>Cxf</td>
<td>0.2861</td>
<td>$-1.4 \times 10^{-2}$</td>
<td>8.6</td>
<td>$&lt;0.00001$</td>
</tr>
<tr>
<td>Derby</td>
<td>0.2537</td>
<td>$-9.3 \times 10^{-3}$</td>
<td>9.3</td>
<td>$&lt;0.00001$</td>
</tr>
<tr>
<td>Lucene</td>
<td>0.3699</td>
<td>$-7.5 \times 10^{-3}$</td>
<td>11.0</td>
<td>$&lt;0.00001$</td>
</tr>
<tr>
<td>Openejb</td>
<td>0.3207</td>
<td>$1.8 \times 10^{-3}$</td>
<td>7.1</td>
<td>$&lt;0.00001$</td>
</tr>
</tbody>
</table>

case with relatively small p-values ($<0.00001$), which indi-
cates the significant synchronization between the two kinds of
activities. Answering Research Question 5. Here, it is ex-
pected that the correlation coefficients tend to 0 in the simu-
lated case because the random mechanism of the null model
makes the synchronous development and simulated email
communication activities independent from each other. In
this experiment, we set the time-window $\xi = 1$ (day) and the curve parameter $\sigma = 10$, and the results are stable to
changes to other appropriate values. Note that correlation
coefficient tends to 0 when $\sigma \to 0$ and tends to 1 when $\sigma \to \infty$, therefore $\sigma$ cannot be chosen too small or too large.

4.4 Synchronous Development and Commu-
nication Are Coupled

We also selected five pairs of developers with the lowest
ratio of C-bursts to E-bursts under the same condition and
at the same time have at least one C-burst. We find that
the associated nine developers are distributed in Ant, Derby,
and Openejb. The pair of developers with the lowest ratio
are from Ant and one of them is an adviser rather than a
committer at most time, as described on the Ant webpage:
He has been involved non-stop with the Ant user community
...... He is opinionated, always striving for the best possible
design. His role is also reflected in the content of his emails,
one of which is selected as follows.
4.4.2 Measuring Communication Tool Effects

The significant synchronization between synchronous development and email communication indicates that there are a considerable number of C-bursts that overlap with or closely follow E-bursts. For a C-burst between two developers, denote by $t_C^*$ and $t_C^*$ its start and end time, respectively; then it is called Email-inspired C-burst if we can find an E-burst between this pair of developers with its start and end time equal to $t_E^*$ and $t_E^*$, respectively, satisfying that the time intervals $[t_C^* - \xi, t_C^*]$ and $[t_E^*, t_E^*]$ overlap. For a given project, denote by $N_C^{\text{Real}}$ and $N_C^{\text{Inspired}}$ the average numbers of C-bursts and Email-inspired C-bursts between pairs of developers in the real case, and by $N_C^{\text{Simulated}}$ the average number of C-bursts between the corresponding pairwise developers in the simulated case. Then, the relative email usage when developing code with each other synchronously can be measured by

$$P_E = \frac{N_C^{\text{Inspired}}}{N_C^{\text{Real}} - N_C^{\text{Simulated}}}.$$  \hspace{1cm} (10)

We find that developers are more likely to use email to coordinate with each other in Cxf and OpenEjb, while the usage of email is lower in Cxf and OpenEjb, as shown in Figure 4.

![Figure 4: Email use during synchronous development.](image)

One of the reasons may be that the latter two projects provide other communication tools for users conspicuously. For example, Cxf provides the link to the Chinese language CXF User’s List available at Google Groups on its website, and OpenEjb has a communication group on Twitter (@OpenEJB). However, the higher ratio of email communications adopted by Derby and Lucene doesn’t make the developers in these two projects easier to generate more lines of codes when developing synchronously, as indicated by Figure 1. This phenomenon is interesting and suggests that, comparing with the other more real-time communication tools, email may not be the best choice to coordinate the work between developers.

Note that here the value of $P_E$ may be larger than 1 if developers frequently communicate with each other by email and the time-window is large enough. If there are several alternative communication tools, and assuming one can get all the communication records, this metric can be used to measure the tendency of developers to use these tools to coordinate their work with others.

5. THREATS TO VALIDITY

There are a number of threats to this study. The six OSS projects are selected from the same foundation and are all written in java, which may limit the generalization of the results. The methods therefore need to be tested on a greater variety of OSS projects in the future.

We use LOC per commit, rather than per unit time, to measure the code contribution of developers as it relates to synchronous development, because it is much harder to know the exact amount of time that developers took to write those lines. We acknowledge that LOC is more about the effort of developers than the code quality. Ideally, it is better to use alternative measurements, e.g., the development time of tasks [24] and the number of merge conflicts, to check the benefits of synchronous development, however, such measurements are relatively more difficult to obtain.

We believe that as we defined them, code growth, $\Delta L$, together with effort, $\Delta W$, are suitable to measure the effects of synchronous development given the current data sets, especially since LOC has been used extensively to measure the productivity of developers. Thus, LOC per commit is a proxy of code contribution per developer. Our results indicate that developers indeed become more efficient when they synchronously collaborate with each other, i.e., contribute to more code growth with less effort, which is well supported by the clues found in their emails that developers reminded each other more frequently to expand the current work and make fewer changes to the original code when they develop synchronously.

It may be argued that the code base grows more during C-bursts because developers collaborate more on code that is of significant size or complexity, e.g., adding new features, while cleaning up code may not require synchronous development, hence more lines get deleted in solo mode. However, this assumption is not simple to validate.

Another threat is that we only consider email communication here, while in fact developers may coordinate their work via other communication tools. Such incompleteness of data doesn’t influence the results on RQ1, RQ2, and RQ3, since they only involve commit activities. However, it may indeed influence the results on RQ4 and RQ5 in the following several aspects. For RQ4, the slope of the linear relationship between the numbers of C-bursts and E-bursts, and the variance of the original data, as shown in Figure 3, may be overestimated, since more E-bursts can be expected when considering more communication records and developers may have different preference to use different communication tools. For RQ5, the positive correlation between C-curves and E-curves may also be overestimated when developers are more likely to communicate with each other by email when collaborating on same files synchronously and by other tools otherwise.

6. RELATED WORK

**Collaboration in Socio-Technical Systems** Two individuals are collaborating if they contribute to the same component of a particular socio-technical system. E.g., collaboration among scientists is mainly reflected in their co-authored papers [39, 40], while among musicians or actors in their joint performances [18, 49].

Collaboration in OSS projects has different levels of meaning due to the hierarchical structure of the artifact. Develop-
ers can be considered to collaborate when they have simply worked on the same OSS projects [26, 42]. On the other hand, if we are interested in a particular project and hope to observe its generative process more carefully, the collaboration between developers can be seen in the co-commit activities on same files in this project [46]. There may also be other, more general collaborative activities, such as sharing programming knowledge through communication tools not officially documented in the project [33], but such contributions are difficult to quantify.

In most of these studies, developers are considered to collaborate with each other all the time if they contribute to the same projects or commit to the same files. In contrast, here we allow that developers collaborate with each other at some time and work alone at other times, and quantify those using C-bursts.

**Synchronization in Nature and Society** Synchronization is a very common phenomenon in nature, underlying most of the coordination processes on which biological systems rely, e.g., fireflies that flash in unison [37] and neural activities in cognitive processing [17]. Recently, similar synchronization mechanisms were used to explain the formation of collective opinion [44] and the mimicry of online user actions [10]. As pointed out by Choudhury et al., understanding social synchrony can be helpful in identifying suitable time points for intervention, e.g., viral marketing [10].

One of the most remarkable features of OSS projects is that voluntary developers can work on the same artifacts remotely, which presents extra challenges for the coordination between them due to the lack of face-to-face communication [23]. Recently, real-time remote communication tools, such as Twitter, has been used to coordinate or synchronize the work of developers, meanwhile, OSS sites like Advocato [32] and GitHub [14] already provide user profiles so developers can better understand the current focus of others, and, thus, enable them to synchronize their commits.

Social individuals benefit from synchronous behaviors [53]. For example, synchronized flashing helps male fireflies find females more frequently [8], while synchronized nonverbal cues can influence face-to-face communication in a positive manner [29]. Dabbish et al. suggested that awareness of others’ work can increase the efficiency of coordination among developers, using case interviews [14]. On the other hand, Herbsleb et al. [24] described coordination as a distributed constraint satisfaction problem (DCSP) and found that dense constraints, in terms of dependencies between developers or between software components, slow down development, but don’t significantly affect productivity. In this paper, we define synchronous development between developers and find that developers indeed benefit from synchronous development in terms of code contribution.

**Communication in Synchronization Processes** It is well-known that communication plays an essential role in synchronization processes [1], i.e., individuals can acquire the current states of their neighbors via communication, and adjust their own states correspondingly. OSS projects blanket adopt or recommend for adoption to their participants communication tools like email, Facebook, Twitter, etc. [51]. Some researchers even advocate for the creation of Software Immersion Environments where project artifacts are arranged in physical 3D space [15], so that developers can communicate and coordinate more directly, like in a real world face-to-face communication.

There are still debates on the effect of communication on working efficiency. For example, Herbsleb and Grinter [21] found that lack of communication between software developers introduced more coordination problems. Gutwin et al. [20] supported this viewpoint and suggested that distributed developers do need to maintain awareness of one another by interviews and analyzing email messages in three successful OSS projects, i.e., NetBSD, Apache httpd, and Subversion. On the other hand, however, it is also argued that social communication have some negative effect on the efficiency of work-related activities [35], since both communication and working activities may compete for the time resources of individuals [36, 56]. As a result, a large number of companies have begun monitoring the usage of emails or office telephones, and employees face consequences for misusing them [11]. There is research that focuses on the design of large systems with smaller communication overhead [2].

Communications are also used to build social networks in OSS projects [5, 7, 48], evoking the studies to identify the relationship between social structure and code properties [3, 55]. In this paper, we reveal strong correlation between communication and synchronous development, which can help to better understand when and how much communication is necessary for efficient code development in OSS.

7. CONCLUSIONS

In summary, we find that synchronous development is far more than a chance occurrence and plays an important role in the development of OSS projects. We identified and quantified the positive effect of synchronous development on the contributions of developers and quantified its strong correlation with email communication activities. Thus, this work can help to better understand the role of communication in the development process. Moreover, we also provide a plausible metric to characterize the preference of developers to use a particular communication tool to coordinate their work. The contrast between larger project growth and smaller LOC effort expended during synchronous collaboration is an interesting finding, which can influence the way distributed development is understood and conducted.

This work can be expanded in several ways. First, synchronous development can be generalized beyond co-commit to the same files at close time, to include functionally related files too, based on the dependence relationship between them. Second, larger-scale synchronous developments can be revealed by similar methods since we expect that, in many cases, more than two developers are needed to solve more complex programming problems. Such a synchronous development network can provide more information about the organizational structure and then help to better understand the collaboration mechanisms in large systems. Third, alternative measurements, such as development time, the number of merge conflicts, and the number of bugs, can be used to measure the effects of synchronous development on coordination efficiency more comprehensively.

8. ACKNOWLEDGMENTS

The authors gratefully acknowledge support from the Air Force Office of Scientific Research, award FA955-11-1-0246. QX acknowledges support from the National Natural Science Foundation of China (Grant No. 61004097, 61273212) and the China Scholarship Council (CSC).
9. REFERENCES


[34] 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


