Who is going to Mentor Newcomers in Open Source Projects?

Gerardo Canfora[1], Massimiliano Di Penta[1], Rocco Oliveto[2], Sebastiano Panichella[1]
[1]University of Sannio, Via Traiano, 82100 Benevento, Italy
[2]University of Molise, Contrada Fonte Lappone, 86090 Pesche (IS), Italy
canfora,dipenta@unisannio.it, rocco.oliveto@unimol.it, spanichella@unisannio.it

ABSTRACT

When newcomers join a software project, they need to be properly trained to understand the technical and organizational aspects of the project. Inadequate training could likely lead to project delay or failure.

In this paper we propose an approach, named Yoda (**Y**oung and newc**O**mer **D**eveloper **A**ssistant) aimed at identifying and recommending mentors in software projects by mining data from mailing lists and versioning systems. Candidate mentors are identified among experienced developers who actively interact with newcomers. Then, when a newcomer joins the project, Yoda recommends her a mentor that, among the available ones, has already discussed topics relevant for the newcomer.

Yoda has been evaluated on software repositories of five open source projects. We have also surveyed some developers of these projects to understand whether mentoring was actually performed in their projects, and asked them to evaluate the mentoring relations Yoda identified. Results indicate that top committers are not always the most appropriate mentors, and show the potential usefulness of Yoda as a recommendation system to aid project managers in supporting newcomers joining a software project.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Programming teams.*

Keywords

Developer Mentoring, Mining Software Repositories, Empirical Studies.

1. INTRODUCTION

In the Star Wars imaginary Universe, Yoda[1] is known to have trained a large number of young Jedi (*youngling*). Such a skill is equally important in software projects, where training new developers is a crucial activity. When a newcomer joins a project, she needs to be trained from many different points of view, such as project architecture and implementation details, development guidelines, and organizational aspects. Training is often performed by one or more *mentors* that, during the early stages of project participation for a newcomer, help her in the work and actively discuss with her project details. Once the newcomer has been trained, she can continue to work autonomously. A relatively similar process may occur in open source projects, where, in most cases, the interaction between the mentor(s) and the newcomer occurs through electronic means, e.g., mailing lists or issue tracking systems. A newcomer would likely first start participating to discussion actively, and then would gradually start to commit changes in the source code repository. Previous studies surveying software projects indicated that mentoring of project newcomers is highly desirable [14]. Differently from the problem of bug triaging—which requires to determine a developer who solved in the past a similar task [2]—mentoring requires to find developers having the ability to effectively communicate and train other people. Hence, *a good mentor is someone that (i) has enough expertise about the topic of interest for the newcomer, and that (ii) demonstrated to have enough ability to help other people.*

In this paper we propose Yoda (**Y**oung and newc**O**mer **D**eveloper **A**ssistant) an approach to identify likely mentors in software projects by mining data from software repositories, and to support the project manager in recommending possible mentors when a newcomer joins a project. Yoda is inspired by the *ArnetMiner*[2] search engine for academic researchers in computer science, which identifies relations between students and advisors, based on a series of heuristics [20]. Shortly, given a pair of researchers *Jim* and *Alice*, *ArnetMiner* suggests that *Jim* could have been the mentor of *Alice* based on four factors: f_1: *Jim* and *Alice* published together a large number of papers, f_2: *Jim* has more publications than *Alice*, f_3: *Jim* is older than *Alice* in terms of activity, and f_4: *Jim* and *Alice* started to publish together as soon as *Alice* started her activity. Even if the student-advisor relationship in academia is one-to-one, the metrics employed by *ArnetMiner* allow to also identify one-to-many relationships.

To identify candidate mentors in the past history of a software project, we consider as potential indicators on mentor-

[1]http://starwars.wikia.com/wiki/Yoda

[2]http://arnetminer.org

ship cases where (i) the mentor and the newcomer exchange a large number of emails in the first phases of the newcomer activity, (ii) the mentor has performed a larger activity in the project than the newcomer in terms of exchanged emails, (iii) the mentor has a longer experience in the project than the newcomer, (iv) the mentor and the newcomer start to exchange emails as soon as the newcomer joins the project, and (v) the mentor is very active in terms of performed commits. The set of indicators above defined is inspired to the *ArnetMiner*, although metrics being used are different: for example, co-authorship is replaced by level of communication; plus, differently from *ArnetMiner*, Yoda needs to account both communication activity (measured in terms of exchanged emails) and technical activity (measured in terms of commits). Once being able to identify likely mentors in the past history of a software project, Yoda recommends mentors each time a newcomer joins the project, by selecting, among the candidate mentors previously identified, those who discuss similar topics to what the newcomer is discussing.

We have evaluated Yoda on data from five open source projects, Apache httpd, the FreeBSD kernel, PostgreSQL, Python, and Samba. First, we investigated which factors or combination of factors provide more accurate indicator of mentoring. Then, we evaluated the accuracy of Yoda when recommending mentors for a newcomer. Finally, we also present the results of a preliminary survey conducted with some people involved in the suggested mentoring relationships of the five projects to understand (i) whether what we identified was actually true, and (ii) to understand how mentoring was performed (and if it was performed) in that project.

The paper is organized as follows. Section 2 describes Yoda, the approach proposed to identify and recommend mentors. Section 3 describes the empirical study aimed at evaluating Yoda on the five open source projects. Section 4 reports the study results. Section 5 provides a qualitative discussion of the empirical study performed, while Section 6 discusses the limitations and threats to validity of both Yoda and of the study. Finally, Section 8 concludes the paper after a discussion of the related literature (Section 7).

2. HOW TO IDENTIFY MENTORS

To recommend mentors, Yoda first identifies a list of available mentors by mining mailing lists and versioning systems. Then, it analyzes the requests for help issued by a newcomer and identifies a list of candidate mentors that could help her. The next sections detail the approach we use to (i) identify mentors from historical data of existing projects; and (ii) to recommend mentors when a newcomer joins a project.

2.1 Who could be a good mentor?

To identify mentors in software projects, we define factors inspired by the ones *ArnetMiner* uses. However, we replace the activity of co-authoring a paper with the activity of exchanging emails and, in addition, we consider the technical activity of a candidate mentor, in terms of number of commits:

- f_1: captures whether, after a newcomer joins a project, she mainly collaborates with a specific person. We measure this as the percentage of emails a newcomer exchanges with an older project member in the first period (we set this to one month) after joining the project, out of the total number of emails exchanged in that period.
- f_2: captures the difference of the amount of discussion activity carried out by mentors and newcomers in terms of the number of emails exchanged.
- f_3: captures the "age" difference between the newcomer and the likely mentor, i.e., the difference (in months) between the date when the newcomer exchanged her first email in the project and the date when the likely mentor did.
- f_4: tells whether the mentor is one of the people the newcomer starts to collaborate first. It is defined in terms of time difference (in months) between the first email exchanged by a newcomer, and the first email the newcomer exchanged with the likely mentor.
- f_5: considers the number of commits performed by a candidate mentor, as an indicator of the past *technical* activity performed by the candidate mentor.

To allow aggregating the five factors, we normalize them in the interval $[0 \ldots 1]$, and then aggregate them into a score defined as

$$\sum_{i=1}^{5} w_i \cdot f_i \qquad (1)$$

where w_i is the importance (weight) attributed to f_i. Determining appropriate weights w_i for f_i—and, as a consequence, also whether each of the f_i is considered or not—is part of the empirical evaluation and will be detailed in Section 3.1. By using the score defined above, it is possible—from the past history of a project—to rank all other developers with the aim of identifying the list of available mentors that can be suggested to newcomers. The higher the rank, the higher the likelihood that the developer played the role of mentor for the considered newcomer. Once the developers have been ranked, a threshold is used to cut the ranked list and identify the top related developers that represent the candidate mentors.

Defining a "good" threshold *a priori* to cut the ranked list is challenging, because a newcomer could have one or more mentors. Thus, we use a scaled threshold t—used in previous work on traceability recovery [1]—based on the values of the factors computed considering the newcomer i and the top developers in the ranked list $t_i = \lambda \cdot TOP_i$, where TOP_i is the value of the factor between the newcomer i and the top developer in the ranked list, while $\lambda \in [0, 1]$. The defined threshold is used to remove from the ranked list developers that have a factor value lower than $\lambda\%$ of the factor value between the newcomer and the top ranked developer.

2.2 Building the project committers' communication network

We identify communication between developers by analyzing mailing lists and by linking names/email in mailing lists to committer IDs extracted from the versioning repository. The link to versioning repositories is needed since we are interested to select people who committed changes to the project repository.

Given two mailing list contributors *Tom* and *John*, we assume a social interaction between them if the mailing list contributor *Tom* sends at least one email to *John* through the mailing list. For the purpose of our study, other than just extracting the network, we also need to keep track of all messages exchanged by mailing list contributors over time. Mailing list contributors and their social interactions are extracted from mailing list messages by identifying the sender name and email address and, whenever available, the recipient/reply-to name and email address. Clearly, since messages can be sent to a mailing list rather than to single addresses, we assume that (i) *Tom* and *John* communicate if there is a direct message with *Tom* as sender and *John* as recipient or *vice versa*, or (i) if *John* answers, even if replying to the mailing list to a thread initiated by *Tom*, or to which *Tom* has previously replied.

When building the contributors' network, we need to unify the names belonging to the same person, i.e., cases of people referring themselves differently, e.g., *John Fitzgerald Smith*, *John F. Smith*, *John Smith*, or *J. Smith*. We use an approach similar to the one proposed by Bird *et al.* [6] that we also applied to analyze mailing lists for a different research [10]. We use similar heuristics to map versioning system IDs to mailing list names/emails, which could be straightforward in cases where the committer ID is a full name and/or an email (e.g., in projects adopting Git as versioning system), while it is less obvious when short IDs are used (such in the case of many CVS repositories). Specifically, we use heuristics to match IDs composed of initials to contributor names—e.g., mapping, where this does not induce ambiguities, *jfk* to *john fitzgerald kennedy*—or finding mailing list contributor names to be linked to committer IDs like *johnsmith* or *jsmith*, trying to compose mailing list contributors first and last names, or first/middle name initials and last name. Further details can be found in a previous paper by Canfora *et al.* [10].

2.3 Recommending mentors

When a contributor joins a project, she needs to become knowledgeable on a specific (set of) topic(s). The project manager needs to find a mentor for this newcomer, that is a contributor that (a) demonstrated already to be a mentor in the past, and (b) actually worked/discussed topics related to those the newcomer is going to work. As for requirement (a), we use the method defined in Section 2.1 to identify available mentors in the past project history. Turning to requirement (b), we need to select among the available contributors that demonstrated to be mentors in the past, the ones that had enough expertise on the topic(s) which the newcomer is going to work on. Rather than proposing a novel approach to identify the most appropriate mentor for a given topic, we use approaches similar to what proposed in the past for bug triaging [2, 9, 19], which proposed to assign a bug to a developer that in the past worked on a bug having a (textually) similar bug report.

Specifically, let us suppose a newcomer p joins the project at time t_x, and let us consider the set of n mentors $M = \{m_1, \dots m_n\}$ identified using the method described in Section 2.1 in the period before t_x. We instantiate an Information Retrieval (IR) [5] process to rank the available mentors, where each document d_i with $i = 1 \dots n$ consists of the union of the text of all emails exchanged by the mentor m_i before t_x, while the query q_p is represented by a request for help submitted by the newcomer p.

In a live setting—when Yoda would be instantiated as a tool—the request for help can be explicitly submitted by the newcomer when she joins the project. For the purpose of our empirical study, since we are working on past data from open source projects, we simulated the query by considering the union of the emails sent by the newcomer during the first week when she joined the project, i.e., within one week after the first email. We consider a week as a time span on the one hand long enough for a newcomer to send emails and precisely ask what kind of help she needs (e.g., taking the first email only could be considered too vague), on the other hand not so long to have an unrealistic scenario.

Both documents and query are processed by removing English stop words, performing stemming using Porter stemming [16], and indexing the text corpus using word frequencies. We use raw frequencies (*tf*) [5] rather than the widely used *tf-idf* [5] as our aim is to mach similar corpus rather than to discriminate different documents, reason for what *tf-idf* results useful. Then, we compare the query q_p with each d_i using the asymmetric Dice coefficient [5]:

$$Dice_p = \frac{\sum_{j=1}^{N} tf_{j_{q_p}} \cdot tf_{j_{d_i}}}{\sum_{j=1}^{N} tf_{j_{q_p}} \cdot tf_{j_{q_p}}}$$

where N is the size of the vocabulary of all words contained in our documents, $tf_{j_{q_p}}$ and $tf_{j_{d_i}}$ are the raw frequencies of the j^{th} dictionary term in the query and d_i, respectively. In other words, the asymmetric Dice coefficient captures how much text of the newcomer's request for help is covered by the mentor's emails. We use it instead of the *cosine* coefficient because the Dice coefficient does not penalize mentors having email corpus that contains much text not contained in the newcomer's request.

3. EMPIRICAL STUDY DEFINITION

The *goal* of this study is to evaluate the performances of Yoda in identifying mentors in software projects, and in recommending mentors for project newcomers. The *quality focus* is concerned with the capability to precisely identify and recommend mentors that (i) had a mentoring experience in the past and (ii) have discussed topics related to the newcomer needs. The *context* consists of discussions extracted from mailing lists and changes extracted from versioning systems for five open source projects, namely Apache httpd, the FreeBSD kernel, PostgreSQL, Python, and Samba.

The five projects are different in terms of size and application domain. Apache httpd is an open-source HTTP server, FreeBSD is a free, open source Unix operating system, PostgreSQL is a database management system (DBMS), Python is a well-known scripting language, and Samba a cross-operating system layer for printer and file sharing. The top part of Table 1 reports key information about the five projects and, above all, of their mailing lists and versioning systems,

Table 1: Characteristics of the five projects analyzed, and of the training and test sets for evaluating Yoda.

Variable	Apache httpd	FreeBSD	PostgreSQL	Python	Samba
Project URL	http://httpd.apache.org	http://www.freebsd.org	http://www.postgresql.org	http://www.python.org	http://www.samba.org
Period considered	08/2001-12/2008	11/1998-10/2008	10/1998-03/2008	05/2000-12/2008	04/1998-12/2008
Size range (KNLOC)	271-850	3,552-7,853	223-522	464-683	156-1,157
Mailing list contribs (Mc)	6,726	23,872	2,935	20,827	3,411
Committers (Cm)	108	640	34	147	229
$Mc \cap Cm$	66	393	29	147	226
Emails exchanged by $Mc \cap Cm$	135,243	2,246,425	44,244	7,479	19,073
Period (training set)	08/2001-03/2002	11/1998-02/2000	10/1998-05/2001	05/2000-05/2001	04/1998-09/2000
Period (test set)	04/2002-12/2008	03/2000-10/2008	06/2001-03/2008	06/2001-12/2008	10/2000-12/2008
# of mentors (training set)	19	65	10	28	17
# of newcomers (training set)	13	33	8	32	33
# of newcomers (test set)	13	33	7	31	33

namely project URL, time period analyzed, size range in KLOC, number of mailing list contributors (Mc), number of committers in the versioning systems (Cm), their intersection ($Mc \cap Cm$), and number of emails exchanged by $Mc \cap Cm$, i.e., by the set of project contributors we considered.

3.1 Study Procedure

The study addresses the following research questions:

- **RQ1:** *How can we identify mentors from the past history of a software project?* This question aims at investigating whether particular combinations of the factors f_1–f_5 described in Section 2.1 can be used to identify mentors, and how good are such combinations compared with some baselines, e.g., using as mentors the project managers or the top project contributors.
- **RQ2:** *To what extent would it be possible to recommend mentors to newcomers joining a software project?* This research question investigates how accurately could Yoda recommend a mentor—among those identified in **RQ1**—for a newcomer that joins the project and is willing to work on a certain topic, and how such a prediction compares with a baseline alternative, i.e., suggesting owners/top committers of artifacts traced to the first emails the newcomer sends when joining the project.

To address **RQ1**, we use different combinations of f_1–f_5 to identify—for each newcomer joining a project—a ranked list of candidate mentors, and then we evaluate them manually. It is worthwhile to recall that such a metrics combination suggests a ranked list of candidate mentors for each newcomer by observing the past project history, without however requiring a training set (the approach is unsupervised) and that, in the context of **RQ1** we are only interested to see whether the f_1–f_5 metrics can identify good mentors, without checking whether the mentor has appropriate skills required by the newcomer (we deal with such an issue in **RQ2**).

First, we perform a calibration of the weights in equation (1). We consider different possible combinations of f_1–f_5, i.e., all ones with equal weight, single factors alone, all possible pairs, all groups of three, and four factors. It is important to note that, by considering f_5 (normalized number of commits) alone, we provide a sort of baseline for our technique, because the number of commits is a metric often used to identify experts—or at least code ownership [8]—in software projects [17].

Then, we give varying weights to the five factors, with the aim of investigating whether there are factors that are more important than others. For example, for combinations of three factors, say f_1, f_2, f_3, we consider, other than the combination with equal weights, i.e., $0.33 \cdot f_1 + 0.33 \cdot f_2 + 0.33 \cdot f_3$, the following cases: (i) one factor weighted more than the others, e.g., $0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_3$, and similarly for f_2 and f_3; and (ii) two factors weighted more than the other, e.g., $0.4 \cdot f_1 + 0.4 \cdot f_2 + 0.2 \cdot f_3$, and similarly for (f_1, f_3) or (f_2, f_3).

To validate the candidate newcomer-mentor pairs, one of the authors and another PhD student (not aware of how Yoda works) manually inspected (independently, discussing cases where they disagreed) the communication between the newcomer and the mentor. The pair mentor-newcomer is classified as a *true positive* if there is a clear evidence of cases in which the newcomer asked help to—and received help from—the mentor, otherwise it is classified as a *false positive*.

To address **RQ2**, we split the project history into two periods, a first period (*training set*) in which we identify the set M of people who have been identified as candidate mentors using the most suitable combination of factors determined in **RQ1**, and a second period (*test set*) for which we aim at recommending mentors for project newcomers. For each project, we have chosen training and test set in order to have a balanced number of newcomers in the two sets (see the bottom part of Table 1).

For each newcomer i identified in the test set, we identify the top k mentors ($m_j \in M$) that exhibit the most similar discussion to the newcomer request for help. For each possible newcomer, we produce a ranked list of k recommended mentors[3], as in realistic scenarios it can happen that the most suitable mentor is not available, and one has to choose the second-best, the third-best, and so on. Since in the context of a post-mortem analysis of data from open source projects like the one we are doing we do not have available other sources of requests for help, we assume that the topic on which a newcomer requires helps is contained in the first emails she exchanges in the project. We define below, in the study settings, how the number of emails has been chosen. Then, for each newcomer i in the test set, we identify mentor(s) using the approach described in Section 2.3, manually validate it, compare with the recommended mentor, and report the number and percentage of pairs for which the recommendation was correct or incorrect.

[3]In Section 4.2 we report results for $k = 1$ and $k = 2$, while the Appendix also reports results for $k = 3$ and $k = 4$.

In addition to such a manual validation, we consider—also for **RQ2**—a baseline, i.e., an alternative way of suggesting experts. To this aim, we trace the textual corpus of the newcomer early emails onto source code files, using an approach proposed by Bacchelli *et al.* [4]. Briefly, such approach aims at matching file names (or class names) in the email text. In some cases (37 out of 117 times in our study) the mapping is straightforward because the emails contain a file name, including extension and even in some cases a complete path. In other 14 cases, the email only contained one or more words that were also file names, which could lead to possible imprecisions (e.g., the word "main" might not refer to a file). To deal with this imprecision, the resulting 37+14=51 traceability links were also manually validated. After that, we took the list of top committers for the files traced onto the emails, and consider such a list as an alternative way of recommending mentors.

3.2 Surveying project developers

As a further evaluation, after having identified candidate pairs of mentor-newcomer, we surveyed these developers by sending them an email, explaining them the purpose of our research activity, and asking them to complete an on-line survey questionnaire—built using *SurveyMonkey*[4] (an example of which is reported in the Appendix)—asking questions about: (i) the experience as project contributor, and whether the contributor is still active; (ii) whether the person mentored a project newcomer and, if yes, how important was the mentoring perceived; (iii) whether the person was mentored when she joined the project, and how important was the mentoring in the decision to stay in the project; and (iv) which are the characteristics of a good mentor (e.g., experience, communication skills, project knowledge).

In addition, the survey page showed to respondents a list of people that Yoda highlighted in the mentor-newcomer candidate pairs, asking them to tell, for each person, whether the respondent (i) was actually the person indicated there, (ii) was a mentor for that person, (iii) was advised by that person, or (iv) never got in touch with that person. We invited to the survey 23 developers from Apache httpd, 37 from FreeBSD, 15 from PostgreSQL, 27 from Python, and 37 from Samba. The questionnaire was completed by 6 developers from Samba, 3 from FreeBSD, 2 from PostgreSQL, and 1 from Python.

4. RESULTS

This section reports the results of the empirical study. The experimental package (with raw data) is available for replication purposes[5].

4.1 RQ1: How can we identify mentors from the past history of a software project?

Figure 1 and Table 2 report, for the five projects we analyzed, the performances of Yoda in detecting mentors for all newcomers joining the project during the entire analyzed time interval. In each subfigure of Figure 1, the x-axis indicates the number of mentors that the approach can recommend when achieving a precision shown on the y-axis.

[4]www.surveymonkey.com
[5]http://www.rcost.unisannio.it/mdipenta/mentoring-data.tar.gz

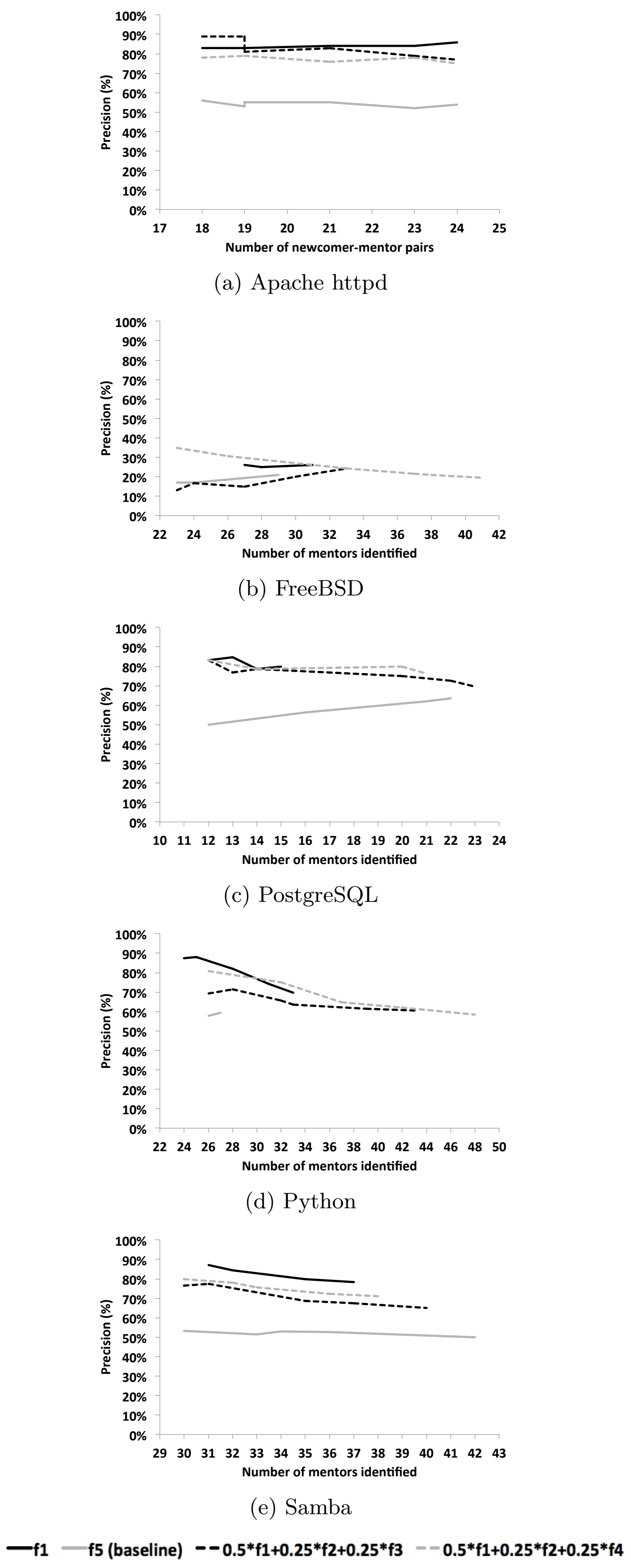


Figure 1: Mentor identification performances for the best combinations of f_1–f_5 and for the baseline (f_5).

Table 2: Precision (%) and number of newcomer-mentor pairs identified for different values of λ.

System	Formula	$\lambda = 100$		$\lambda = 90$		$\lambda = 80$		$\lambda = 70$		$\lambda = 60$		$\lambda = 50$	
Apache httpd	f_1	83%	18	83%	18	83%	18	84%	19	84%	19	86%	21
	$f_5(baseline)$	56%	18	53%	19	55%	20	55%	20	52%	21	54%	26
	$0.33 \cdot f_1 + 0.33 \cdot f_2 + 0.33 \cdot f_3$	83%	18	85%	20	82%	22	83%	24	80%	25	77%	26
	$0.33 \cdot f_1 + 0.33 \cdot f_2 + 0.33 \cdot f_4$	78%	18	75%	20	77%	22	77%	22	77%	22	78%	23
	$0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_3$	89%	18	89%	18	81%	21	83%	23	79%	24	77%	26
	$0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_4$	78%	18	79%	19	79%	19	76%	21	78%	23	75%	24
FreeBSD	f_1	26%	27	26%	27	26%	27	26%	27	25%	28	26%	31
	$f_5(baseline)$	17%	23	17%	23	17%	23	17%	24	17%	24	21%	29
	$0.33 \cdot f_1 + 0.33 \cdot f_2 + 0.33 \cdot f_3$	17%	23	17%	24	17%	24	17%	24	19%	26	23%	31
	$0.33 \cdot f_1 + 0.33 \cdot f_2 + 0.33 \cdot f_4$	30%	23	25%	28	20%	35	20%	35	21%	39	20%	40
	$0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_3$	13%	23	13%	23	17%	24	15%	27	20%	30	24%	33
	$0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_4$	35%	23	31%	26	24%	33	22%	37	21%	38	20%	41
PostgreSQL	f_1	83%	12	85%	13	79%	14	79%	14	80%	15	80%	15
	$f_5(baseline)$	50%	12	50%	12	50%	12	56%	16	62%	21	64%	22
	$0.33 \cdot f_1 + 0.33 \cdot f_2 + 0.33 \cdot f_3$	83%	12	76%	17	75%	20	65%	23	62%	26	55%	29
	$0.33 \cdot f_1 + 0.33 \cdot f_2 + 0.33 \cdot f_4$	83%	12	77%	13	73%	15	76%	21	73%	22	70%	23
	$0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_3$	83%	12	77%	13	79%	14	75%	20	73%	22	70%	23
	$0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_4$	83%	12	79%	14	79%	14	79%	14	80%	20	76%	21
Python	f_1	88%	24	88%	24	88%	25	82%	28	74%	31	70%	33
	$f_5(baseline)$	58%	26	58%	26	59%	27	59%	27	59%	27	59%	27
	$0.33 \cdot f_1 + 0.33 \cdot f_2 + 0.33 \cdot f_3$	62%	26	68%	31	66%	32	64%	36	63%	40	61%	41
	$0.33 \cdot f_1 + 0.33 \cdot f_2 + 0.33 \cdot f_4$	73%	26	75%	28	74%	34	63%	40	60%	45	57%	47
	$0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_3$	69%	26	71%	28	66%	32	64%	33	62%	39	60%	43
	$0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_4$	81%	26	81%	26	75%	32	65%	37	62%	42	58%	48
Samba	f_1	87%	31	84%	32	84%	32	80%	35	80%	35	78%	37
	$f_5(baseline)$	53%	30	52%	33	52%	33	53%	34	53%	36	50%	42
	$0.33 \cdot f_1 + 0.33 \cdot f_2 + 0.33 \cdot f_3$	77%	30	72%	32	69%	35	69%	36	66%	38	64%	42
	$0.33 \cdot f_1 + 0.33 \cdot f_2 + 0.33 \cdot f_4$	73%	30	70%	33	68%	34	67%	36	63%	40	63%	41
	$0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_3$	77%	30	77%	31	69%	35	69%	35	68%	37	65%	40
	$0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_4$	80%	30	78%	32	78%	32	76%	33	72%	36	71%	38

As described in Section 3.1, we evaluated the performance of Yoda for all combinations of f_1–f_5, as well as for combinations having varying weights. Due to space limitations, Table 2 only reports a subset of the analyzed combinations (complete tables can be found in the Appendix), and specifically:

- f_1, i.e., the percentage of exchanged emails between newcomer and mentor. We noticed that, if considering each of the f_1–f_5 factors alone, such a factor is the one that produces the best performances.

- f_5, i.e., the normalized number of commits performed by the candidate mentor. We use this factor as a baseline, to determine whether an obvious choice of mentors, i.e., top committers, could be appropriate.

- $0.33 \cdot f_1 + 0.33 \cdot f_2 + 0.33 \cdot f_3$ and $0.33 \cdot f_1 + 0.33 \cdot f_2 + 0.33 \cdot f_4$, because we considered that the combination of f_1 and f_2 with either f_3 or f_4 produces the best results among combinations obtained weighting all factors similarly.

- $0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_3$ and $0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_4$, i.e., again considering f_1 and f_2 with f_3 or f_4, weighting f_1 twice than the other factors. These are the combinations able to achieve the best performances.

Note that Figure 1 contains exactly the same information of Table 2 (except that we do not report the combination of f_1, f_2, and f_3 or f_4 with equal weights since they are less interesting to be compared), and has the purpose of better allowing the comparison of different combinations, whereas Table 2 provides precise figures.

In summary, the obtained results indicate that:

- f_5 (number of commits) is not a good indicator of mentorship. Also, f_5 does not even provide a useful contribution if used in combination with other factors. While it is true that a very active committer can be expert on a particular topic, she might not be very willing (or able) to exchange ideas and/or instruct other people.

- $0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_3$ and $0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_4$ achieve the best performances, with a precision over 70% and in many cases well above 80%. Also, it can be noticed that such a precision does not decrease when decreasing λ, which means that the chosen combinations are able to recommend a wider set of candidate mentors without sacrificing the precision.

- f_1 alone tend to be as precise as the other combinations, and in some cases (Python and Samba) even more precise than the combinations above. However, it is only able to recommend a limited set of mentors even when decreasing λ. Although, in the practice, a single mentor would suffice, having a good recall is desirable because (i) not all candidate mentors may be available when needed, and (ii) among the candidate mentors, we need to select those having the expertise required by the newcomer. Hence, to obtain a good balance between precision and number of mentors (which as said

Table 3: Number and percentage of correct and incorrect top 1 and top 2 mentor recommendations for newcomers in the test set.

	Top 1		Top 2	
System	Correct	Wrong	Correct	Wrong
Apache	11 (85%)	2 (15%)	21 (81%)	5 (19%)
FreeBSD	10 (30%)	23 (70%)	16 (24%)	50 (76%)
PostgreSQL	7 (100%)	0 (0%)	14 (100%)	0 (0%)
Python	20 (65%)	11 (35%)	48 (77%)	14 (23%)
Samba	31 (94%)	2 (6%)	54 (82%)	12 (18%)

above does not mean to accept a very low precision), it is necessary to combine f_1 (having however a higher weight) with f_2 (rewarding mentors in general very active in discussions) and f_3 (mentor/newcomer project age difference) or f_4 (the newcomer mainly collaborate with the mentor in her early stages). In summary, the percentage of exchanged emails *per se* is not enough to identify a large set of candidate mentors.

The results achieved also indicate that on FreeBSD Yoda exhibits low performances (precision around 30%) while for all other systems results are pretty consistent. Our interpretation is that such a result does not depend on the system size and on its number of project contributor. In fact, on other systems having a high number of contributors (such as Python or Samba), Yoda exhibits good performances. We believe that this can be due to the nature of the discussion occurring in the FreeBSD mailing lists. By inspecting the emails we realized that most of the discussion is one-to-many, i.e., people posting issues or suggestions to many other people or to the whole mailing list. Although the way we analyze mailing lists allow to treat—with some approximation many-to-many communications—the FreeBSD communication did not always exhibit dominant persons (potential mentors). This, clearly, makes the identification of pairwise collaborations less precise.

It is possible to identify mentors—with a precision of 70% or above—in the past history of a software project by using the combinations of factors $0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_3$ or $0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_4$. Considering top committers is not a precise and reliable metric to identify mentors.

4.2 RQ2: To what extent would it be possible to recommend mentors to newcomers joining a software project?

To evaluate the recommendation method, we split the history of each project in a training set and test set. The bottom part of Table 1 reports, for each project the time interval of training and test sets, the number of mentors identified in the training set, and the number of newcomers in the training set and test set.

Table 3 reports the accuracy of Yoda in recommending mentors for the newcomers of the test set. We recommend mentors among those identified—in the training set—using $0.5 \cdot f_1 + 0.25 \cdot f_2 + 0.25 \cdot f_3$, which as explained in **RQ1** exhibits in most cases the best performances. The left-side of each column in the table reports the number and percentage of correct and incorrect top 1 mentor recommendations for each newcomer (ranked according to Dice asymmetric similarity), while the right-side reports the number and percentage of correct and incorrect top 2 recommendations.

Table 4: Precision in the recommendation of mentors among top committers of files traced onto newcomers emails.

	Top 1		Top 2	
System	Correct	Wrong	Correct	Wrong
Apache	0 (0%)	6 (100%)	1 (8%)	11 (92%)
FreeBSD	1 (6%)	15 (94%)	1 (3%)	31 (97%)
PostgreSQL	1 (50%)	1 (50%)	1 (25%)	3 (75%)
Python	0 (0%)	7 (100%)	1 (7%)	13 (93%)
Samba	7 (35%)	13 (65%)	14 (35%)	26 (65%)

As the table shows, the percentage of correct top 1 recommendations is very high, above 85% except for Python, where it is 65%, and for FreeBSD, where it is 30% (for the reasons explained in **RQ1**). Even when we consider the top 2 recommendations, the correctness remains high, 71% or above—including this time Python where it increases to 77%—and again with the exception of FreeBSD where it is 24%. In the Appendix we also report results for top 3 and top 4 recommendations, which nevertheless only slightly increase or decrease with respect to top 1 and to top 2. As explained in Section 3.1, results of Table 3 are obtained by considering as newcomer query (to identify recommenders) emails sent during the first week in the project. As shown in the Appendix, results do not substantially vary if considering a longer period (up to four months), which is however less realistic because we cannot expect the newcomer would wait so long to get a mentorship.

For 51 out of the 117 newcomers for which it has been possible to trace the first-week emails onto source code files, Table 4 reports the precision in the recommendation in the same way as for Table 3. As the table shows, the precision is quite low: always below 10%, with the exception of Samba where is 35%, and top 2 for PostgreSQL where is 25%.

Yoda is able to recommend mentors with a correctness of about 65% or above for the top 1 recommendations and 71% or above for the top 2 recommendations for all systems except FreeBSD, where the percentages decrease to 30% and 24%, respectively. Top committers or files related to newcomer early emails cannot be used to recommend mentors.

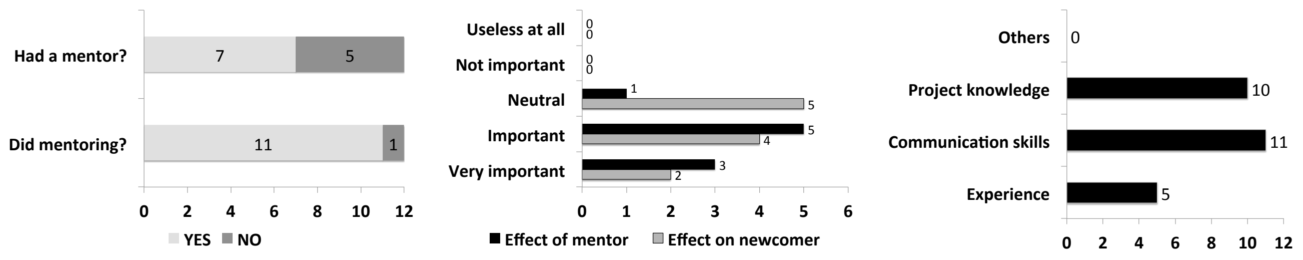
5. DISCUSSION

In the following, we provide additional, qualitative insights to the quantitative study reported in Section 4.

5.1 Hints collected from project contributors

This section reports results we collected surveying project developers. Since we collected a limited number of answers (12), we report aggregate data only for the purpose of explaining how the respondents perceived the importance of the mentoring process.

Figure 2(a) indicates that, out of 12 developers, 11 admitted to have been mentors, and 7 indicated that they were men-



(a) Done/received mentoring (b) Perceived importance of mentoring (c) What makes a good mentor

Figure 2: Survey questionnaire answers: generic questions on mentoring activity and its importance.

tored by someone else. Figure 2(b) reports the perceived importance of mentoring when respondents performed mentoring themselves or when they were mentored. The effect was perceived very important (3) or important (5) for developers performing mentoring, while when developers received mentoring, 2 developers indicated that such activity is very important and 4 that it is important. In summary, developers indicated that mentoring is important, although it seems that developers are more likely to admit that they performed mentoring than they were mentored. Also, mentoring seems to be perceived more important by developers that performed it rather than by developers that received it.

Figure 2(c) reports what factor developers indicated to be important for good mentoring. The respondents suggested it is mainly matter of communication skills (11) and project knowledge (10), while only 5 developers indicated that experience can play an important role. However, one of the respondents said *"Maybe good communication skills is the least important of the three above, if one is working closely together."*, thus the importance of communication skills depends of the cohesiveness of the project team.

Some developers added comments to the survey to indicate that they performed/received mentoring using communication means different from emails. For example, they added comments like *"Suggested work they can do. But mostly over IRC."*, or *"Not only mailing lists, but also IRC and direct communication."*

By classifying developers in the list we provided, the respondents indicated us a set of 23 mentor-newcomer pairs. We identified 10 of them correctly (the remaining 13 were not identified), while we identified 3 pairs for which respondents indicated that mentoring did not occur. Thus, with respect to the answers provided in the survey questionnaire, our approach has a precision of 10/(10+3)=77% and a recall of 10/23=43%.

5.2 Examples of cases where Yoda worked well and where not

In the following we report some examples of collaborations and fragments of emails indicating cases in which Yoda worked well to identify mentors (**RQ1**) and cases where it did not. Cases of *true positives*—confirmed by respondents of our survey—were made evident by exchanges of email in which the newcomer clearly asked for help.

Figure 3 shows an example of newcomer (Bjoern)-mentor (James) network found in Samba, when the newcomer approached the project by sending his first emails (Figure 3(a)), and after one year (Figure 3(b)). Note that edges are labeled with the number of exchanged emails. James had already a three-years experience (with 150 commits performed) when Bjoern approached him. As it can be seen from the network, James has a high degree (=13), and Bjoern mainly exchanges emails (4 emails exchanged) with James rather than with other people. After one year, the social importance of James increase, as well as (to some extent) the one of Bjoern, which however still has James as main contact (18 emails exchanged). The mentoring relationship is also evident from fragments of their communication: *"Hi James, maybe you can bring some light into the dark here: I did some tests with ..."*

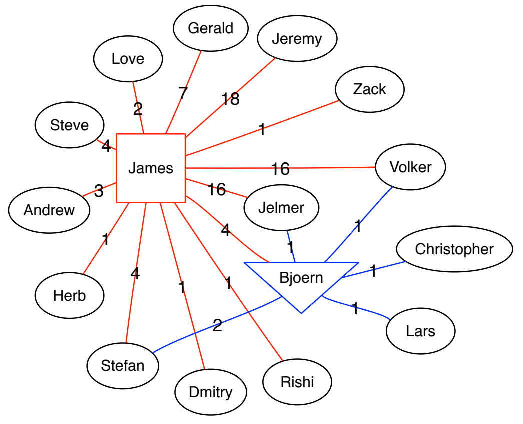
Another example (confirmed by the survey respondent) is the one in Samba where Kai asked Jeremy[6] help because she wanted to contribute to the project *"As you might know, I'm a ... student implementing NTLMSSP signing/sealing in Wine. I've worked on basic NTLM authentication for Wine last year, using ntlm_auth. ... I decided to give Samba4's GENSEC subsystem a try."*. Then, Jeremy responded with a detailed list of instructions answering Kai specific technical questions, e.g. *"This is the quickest way to make this work (IMHO)."*, *"Probably second best solution."* (where Kai proposed two possible implementations), or even discouraging Kai to implement some features *"this will be a long long road to walk..."*

One example of *false negative* is when the newcomer asked for help and the mentor never (or seldom) responded. An instance of such a case was reported as a true mentoring in our survey, however Yoda did not detect it because of the low value of f_1.

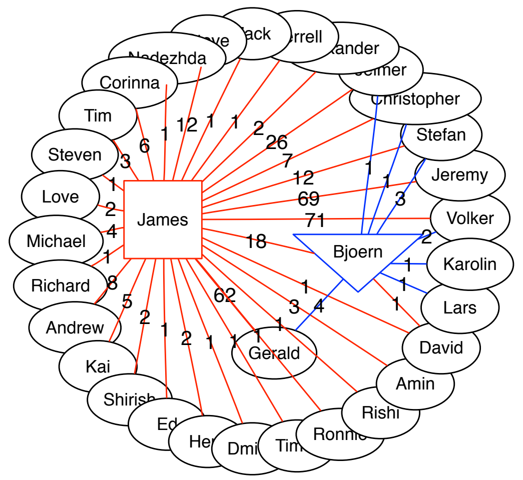
One example of *false positive* concerns the collaboration between Ivan and Robert in FreeBSD. Robert helped Ivan to deal with some performance problem of the SSH protocol implementation. Although also from the email it appeared clear that Robert—indicated as candidate mentor by our approach—was the expert giving suggestions, very likely he helped to solve a specific problem only, so to be considered as someone who helped Ivan, while not really Ivan's mentor.

Now we report hereby one case where mentoring recommen-

[6]We do not report last names for confidentiality reasons.



(a) One week after Bjoern approached the project



(b) One year after

Figure 3: Example (from Samba) of developers' network involving a newcomer (Bjoern) and a mentor (James).

dation (**RQ2**) worked and one when it did not. When Lucien joined the Apache httpd project, he asked information about proxy handling *"I'm translating mod_proxy.xml, and I don't understand what the term worker means . . . "*. Then, after he received information about the meaning of this term, he checked whether he correctly understood: *"So, say we have this proxy configuration: ProxyPass path1 server1, ProxyPass path2 server2 . . . "*. In summary, Lucien seems to be interested to work on tasks related to proxy. Yoda recommended him Joshua as a mentor, which in the past helped someone else (Laurent) on a problem related to proxy: *"ProxyPassReverse only rewrites Location: headers . . . ."*. That is, the email referred several times to the terms *Proxy* and *Pass*, also contained in the early email sent by Lucien.

An example of *false positive* is between Takashi (newcomer) and Joshua (candidate mentor). Both the early emails of Takashi and previous emails of Joshua contained some common terms, such as "3D", contained in XML and HTML fragments, e.g., *"<note type=3D"warning">"* in the early email by Takashi and *"<meta name=3D"generator"/ >"* in previous emails by Joshua. However, the technical content of the emails was not very similar.

6. YODA LIMITATIONS AND THREATS TO VALIDITY

This section describes the limitations of Yoda and the threats to validity of the study we performed. We are aware that Yoda is limited for the following reasons:

- It identifies mentor-newcomer pairs mainly based on their activity on mailing lists. We are aware, however, as also reported in a paper by Aranda and Venolia [3], that developers can communicate also outside mailing lists, e.g., using a chat. One of the developers that responded to our survey *"I've asked questions personally in private chats."* and when we asked if he helped someone else, he answered *"Yes, suggested work they can do. But mostly over IRC."*.
- It does not account for the availability of mentors. However, since Yoda proposes a list of mentors for a newcomer, the project manager can easily assign to the newcomer a mentor that is currently available.
- It matches the expertise required by the newcomer with the expertise of possible mentors by comparing the early emails of the newcomer with all emails of the mentor. However, as we explained in Section 2.3, it is not the purpose of this paper to propose a better method to identify expertise, as others have been proposed already in the past [2, 9, 19].

Threats to *construct validity* concern the relation between the theory and the observation, and in this work are mainly due to the measurements we performed. First, there could be imprecisions in the way we mapped mailing list names with versioning system IDs. However, we manually validated this mapping. In addition, for systems using git (Apache httpd, Samba, PostgreSQL, Python) the mapping is straight-forward in that full names and emails are used as git IDs. The "age" of a developer is computed by observing the first email exchanged, which might have occurred way before (or way after) the person joined the project. Another aspect of our validation that is inherently subject to imprecision is the manual validation of Yoda recommendations. We limited the degree of error-proneness and subjectiveness by having two different people performing the inspection independently.

Threats to *internal validity* concern external factors, we did not consider, that could affect the variables being investigated. The factors we considered, f_1–f_5, are only a partial view of the mentor and newcomer experiences, and of their inter-communication. There can be other factors we did not consider. Future work will be devoted to consider other factors aiming at mitigating such a threat.

Threats to *external validity* concern the generalization of our findings. We have performed our study on data from five different systems belonging to different domains and having different size in terms of code base and number of developers. Further evaluation is however necessary, especially in industrial environments where there can be a tacit knowledge—within an organization, but not encoded in the communications—of who can be a good mentor and who not.

7. RELATED WORK

There have been several researchers investigating what happens when a newcomer joins a software project, and which are the factors for her growth, including mentoring. Our work falls in the general area of those aimed at relating social relations among developers with technical aspects of the project they are working on. For example, Cataldo *et al.* related the communication activity with software quality [12], and developers' productivity [11].

Dagenais *et al.* [14] studied, by surveying 18 IBM developers, what happens when someone moves into a new "project landscape", making for her necessary to get acquitted with the new environment. Among the factors they found important, it is worthwhile to mention the need for early practicing, the availability of feedback for their work, and *the need for getting initial guidance*. The latter is totally in agreement with what we collected from our small survey, and motivates approaches like Yoda. Fronza *et al.* [15] studied how newcomers join agile projects, finding that pair-programming is used to initiate newcomers to a project.

Zhou and Mockus [22] investigated, on three industrial and three open source projects, how the sociality level of a project in a particular moment influences the likelihood for newcomers to become long-term contributors. They found that it is more likely that this happens when the project sociality is low, because senior developers have more time to train the newcomers. Zhou and Mockus concluded suggesting the need for proper training plans for newcomers in open source projects: this requires to identify appropriate mentors. Zhou and Mockus [21] also identified challenges in a software market where offshoring, outsourcing and open source development are increasing fast: understanding cultural differences, analyzing how developers grow their expertise, and *providing tools to facilitate newcomers' learning*, as Yoda does.

A different perspective of developers' growth in software projects was studied by Sinha *et al.* [18] who investigated, on the Eclipse project, how project contributors become committers, and found that this depends on having contributed patches/source code of the project, being active in other open source projects, or being part of the project organization. Also, Bird *et al.* [7] investigated the phenomenon of "immigration" in software projects, explaining the *who, how and when* in the process of providing to newcomers the authority to commit in the project repository.

In summary, while all these works highlighted the need for supporting newcomers when they join a software project, to the best of our knowledge this is the first paper aimed at proposing an approach to identify mentors in software projects and to recommend them to newcomers.

In the past and recent years, other kinds of recommenders have been proposed to support developers—especially junior ones. Among others, Hipikat [13] provides recommendations about components relevant for the current coding context of the developer. As discussed in Section 2.3, identifying mentors also requires the selection of people having specific expertise. This has been done by Anvik *et al.* [2], by Canfora and Cerulo [9], and by Tamrawi *et al.* [19] who proposed approaches—based on machine learning, IR, and fuzzy clustering respectively—for bug triaging, i.e., to determine the most suited developers able to fix an incoming bug.

8. CONCLUSIONS AND FUTURE WORK

Mentoring is particularly important to make software project newcomers knowledgeable of various aspects of the project on which they are contributing, such as technical details, coding guidelines, and organizational rules. As part of our study we contacted contributors to open source projects, one of which mentioned *"My general view is that it is very important that mentor and mentee share at least the mindset and the same passions. My primary mentors have been . . . and they both had a very strong technical background, something I definitively wanted to match."*

This paper proposed Yoda, an approach to identify mentors by relying on historical data from a software project, and then recommend them when a newcomer joins the project. Being inspired by *ArnetMiner*—a tool that analyzes academic collaborations—Yoda identifies mentoring when the newcomer exchanges most of her emails with the mentor, and the mentor has a higher social importance and project age than the mentor. Then, when a newcomer joins a project, her early discussion is used to identify the expertise she is requiring, and Yoda recommends, among the candidate mentors previously identified, those who exhibited a discussion (textually) similar to the newcomer early discussion.

Yoda has been evaluated on data from five open source projects, Apache httpd, the FreeBSD kernel, PostgreSQL, Python, and Samba. Results of the study indicate that, except for FreeBSD, Yoda is able to identify candidate pairs of mentor-newcomer with a precision in most cases higher than 80%, and recommend them with a precision greater than 70%. Instead, relying on top committers as an alternative to Yoda does not produce as accurate results as those of Yoda. Comments collected from project developers indicated us that mentoring is important and depends on project knowledge and communication skills more than on experience.

Work-in-progress is directed towards different directions. We plan to further improve Yoda by considering factors able to better capture the technical skills of mentors. Also, we plan to replicate the study on further projects. Finally, we plan to develop an assessment technique for developers' network aimed at automatically identifying cases—such as the one of FreeBSD in your study—where Yoda is not applicable as such, and if possible identify countermeasures for such scenarios.

Acknowledgments

We would like to thank the contributors of FreeBSD, PostgreSQL, Python, and Samba who responded to our survey. We would also thank Annibale Panichella for his help in the manual validation of the results.

9. REFERENCES

[1] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.

[2] J. Anvik and G. C. Murphy. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Trans. Softw. Eng. Methodol.*, 20(3):10, 2011.

[3] J. Aranda and G. Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. In *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*, pages 298–308, 2009.

[4] A. Bacchelli, M. Lanza, and R. Robbes. Linking e-mails and source code artifacts. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, pages 375–384. ACM, 2010.

[5] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[6] C. Bird, A. Gourley, P. T. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR 2006, Shanghai, China, May 22-23, 2006*, pages 137–143, 2006.

[7] C. Bird, A. Gourley, P. T. Devanbu, A. Swaminathan, and G. Hsu. Open borders? immigration in open source projects. In *Fourth International Workshop on Mining Software Repositories, MSR 2007, Minneapolis, MN, USA, May 19-20, 2007, Proceedings*, page 6. IEEE Computer Society, 2007.

[8] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. T. Devanbu. Don't touch my code!: examining the effects of ownership on software quality. In *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering and 13rd European Software Engineering Conference, Szeged, Hungary, September 5-9, 2011*, pages 4–14. ACM, 2011.

[9] G. Canfora and L. Cerulo. Supporting change request assignment in open source development. In *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC), Dijon, France, April 23-27, 2006*, pages 1767–1772. ACM, 2006.

[10] G. Canfora, L. Cerulo, M. Cimitile, and M. Di Penta. Social interactions around cross-system bug fixings: the case of FreeBSD and OpenBSD. In *Proceedings of the 8th International Working Conference on Mining Software Repositories, MSR 2011 (Co-located with ICSE), Waikiki, Honolulu, HI, USA, May 21-28, 2011, Proceedings*, pages 143–152, 2011.

[11] M. Cataldo, J. D. Herbsleb, and K. M. Carley. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *Proceedings of the Second International Symposium on Empirical Software Engineering and Measurement, ESEM 2008, October 9-10, 2008, Kaiserslautern, Germany*, pages 2–11. ACM, 2008.

[12] M. Cataldo, P. Wagstrom, J. D. Herbsleb, and K. M. Carley. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *Proceedings of the 2006 ACM Conference on Computer Supported Cooperative Work, CSCW 2006, Banff, Alberta, Canada, November 4-8, 2006*, pages 353–362, 2006.

[13] D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth. Hipikat: A project memory for software development. *IEEE Trans. Software Eng.*, 31(6):446–465, 2005.

[14] B. Dagenais, H. Ossher, R. K. E. Bellamy, M. P. Robillard, and J. de Vries. Moving into a new software project landscape. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, pages 275–284. ACM, 2010.

[15] I. Fronza, A. Sillitti, and G. Succi. An interpretation of the results of the analysis of pair programming during novices integration in a team. In *Proceedings of the Third International Symposium on Empirical Software Engineering and Measurement, ESEM 2009, October 15-16, 2009, Lake Buena Vista, Florida, USA*, pages 225–235, 2009.

[16] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[17] G. Robles, J. M. González-Barahona, and I. Herraiz. Evolution of the core team of developers in libre software projects. In *Proceedings of the 6th International Working Conference on Mining Software Repositories, MSR 2009, Vancouver, BC, Canada, May 16-17, 2009*, pages 167–170. IEEE, 2009.

[18] V. S. Sinha, S. Mani, and S. Sinha. Entering the circle of trust: developer initiation as committers in open-source projects. In *Proceedings of the 8th International Working Conference on Mining Software Repositories, MSR 2011, Waikiki, Honolulu, HI, USA, May 21-28, 2011, Proceedings*, pages 133–142. IEEE, 2011.

[19] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen. Fuzzy set and cache-based approach for bug triaging. In *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering and 13rd European Software Engineering Conference, Szeged, Hungary, September 5-9, 2011*, pages 365–375. ACM, 2011.

[20] C. Wang, J. Han, Y. Jia, J. Tang, D. Zhang, Y. Yu, and J. Guo. Mining advisor-advisee relationships from research publication networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 203–212, 2010.

[21] M. Zhou and A. Mockus. Growth of newcomer competence: challenges of globalization. In *Proceedings of the Workshop on Future of Software Engineering Research, FoSER 2010, at the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November 7-11, 2010*, pages 443–448, 2010.

[22] M. Zhou and A. Mockus. Does the initial environment impact the future of developers. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, pages 271–280. ACM, 2011.