Determining Developers' Expertise and Role: A Graph Hierarchy-based Approach

Pamela Bhattacharya   Iulian Neamtiu
Department of Computer Science and Engineering
University of California, Riverside
Email: {pamelab,neamtiu}@cs.ucr.edu

Michalis Faloutsos
Department of Computer Science
University of New Mexico
Email: michalis@cs.unm.edu

***Abstract*—Determining contributors' expertise, role, and individual importance are fundamental for assessing their impact on a software project. Currently-used expertise metrics are agnostic to contributor roles and can lead to incorrect characterizations. To address these issues, we operationalize contributor expertise and role. First, we revisit current expertise metrics and show that their use bundles many different aspects, creating ambiguity. Second, we introduce clearly-defined contributor roles, which capture multiple project facets. Third, we propose a graph model, based on contributor collaborations, that captures the hierarchical structure of the contributor community in a concise yet informative way. We demonstrate the model's usefulness in two ways: (a) for identifying the structure and evolution of contributor interactions; (b) for predicting contributor roles. We substantiate our study using two large open-source projects, Firefox and Eclipse. Our systematic approach clarifies and isolates contributor role and expertise, and sheds light onto the dynamics of contributors within software projects.**

I. Introduction

Our work is motivated by the fundamental need—present in both free [1]–[5] and commercial [6] software projects—to have a quantitative basis for assessing, and characterizing the evolution of, developers' role, contributions, impact and status. In particular, we are interested in providing a quantitative yet intuitive way of answering questions such as: *Who are the most "valuable" (competent, efficient) developers in a software project? Who are the best bug fixers? Who knows best who the right person is to fix a bug? Can the contributions of X and Y be compared? Are contributors promoted on merit? Where does X stand in the contributor hierarchy, and what is the number and nature of X's contributions?*

Specifically, given the evolution history of a project, our goal is to determine and assess the contributions, as well as the expertise, of each developer in the project. While software repositories contain a wealth of data, extracting actionable information on developer role and expertise from these repositories is hard. For example, only larger projects record structured bug activity information, e.g., the list of developers a bug has been assigned to, who has triaged a bug, and who has ultimately fixed it; when projects use separate systems for version control and bug tracking, as Mozilla and Eclipse do, entity resolution is required; and even for open-source projects, privacy concerns might prevent the release of contributors and their access level [7]. All these factors hinder the reconstruction of a complete picture of the nature and specifics of a developer's contributions, which is essential for determining role and expertise.

Previous work: Despite the significant number of studies on assessing contributor expertise, authorship, and ownership in software projects, very few studies have really focused on the problem at hand. Some studies evaluate metrics to deduce expertise [8], [9], which is slightly different from our focus. Others examine expertise as an absolute value [10]–[14] but as we point out, expertise is multi-faceted. Contributor collaboration as a form of social networking has been studied [15]–[20], but extracting a *hierarchy* structure of collaborations has not been investigated so far. We discuss previous work in more detail in Section VII.

Example 1: Simply using metrics for characterizing contributions without regards to role can be misleading, as illustrated next. Consider two Mozilla contributors, D_1 and D_2.[1] Using widely-used metrics, we find that D_1 and D_2 have roughly the same *percentage of bugs fixed from those assigned* to them, with 49.15% and 53.09% respectively. Both D_1 and D_2 have similar *seniority*, having been with the project for 8 and 10 years, respectively. These numbers would lead someone to believe that D_1 and D_2 exhibit comparable expertise and play similar roles in the project. However, upon closer examination, with more refined role definitions, we find that D_1 serves the role of triager (an individual who assigns a new bug to a developer) while D_2 serves the role of patch tester (an individual who reviews and tests patches submitted for fixing a bug). Our approach is able to make this distinction, as we discuss later.

Example 2: Large open-source projects are founded, and operate, on the basis that these projects are meritocracies, with their governing documents explicitly stating this. In the Eclipse project, meritocracy is a guiding principle: *"Eclipse is a meritocracy. The more you contribute the more responsibility you will earn."* [2]. Similarly for Mozilla, the umbrella project for Firefox: *"Leadership roles are granted based on how active an individual is within the community as well as the quality and nature of his or her contributions. This meritocracy is [...] resilient and effective [...]"* [1]. Other projects adopt the same strategy. For example, Ubuntu's Governance document says *"This is not a democracy, it's a meritocracy"* [3]; Debian has a similar charter: *"The Debian Project [...] is a meritocracy with [...] meritocratic key functional positions."* [4]; Apache operates on the same principle [5].

Therefore, accurately assessing developers' contributions is essential for ensuring the operational integrity of such projects; our approach facilitates such assessments.

[1]We withhold actual names for privacy reasons.

Contributions: In this paper, we operationalize (i.e., develop a systematic approach for defining and determining) contributor expertise and role.[2] We start by revisiting the whole expertise assessment process and show we can capture it systematically. Our key novelty is defining roles that capture fundamental software development functions and then building a framework to assess contributions along these roles. Our main contributions can be summarized as follows:

a. Quantifying the inadequacy of current metrics. We revisit previous expertise metrics: we show that each metric captures a local notion of expertise by quantifying a specific development activity (e.g., LOC added) but when put together, they fail to capture a global notion of expertise (Section II-B). The crux of the problem is that these metrics are agnostic to contributor roles, and if we simply combine them, we bundle many different aspects creating a veil of ambiguity.

b. Defining developer roles. We propose to assess expertise and contribution along roles, which eliminates the aforementioned confusion. For example, an expert bug fixer is not necessarily an expert bug triager, but both are equally important for a project. We introduce a set of roles: Patch tester, Assist, Triager, Bug analyst, Core developer, Bug fixer, Patch-quality improver. We also provide ways to define these roles rigorously (Section III).

c. Proposing an intuitive graph-based model of developer contribution named Hierarchical Contributor Model (HCM). HCM concisely represents contributor interactions in a way that captures hierarchy, role and "importance" of contributors (Section IV), which is hard to do with previously-defined expertise metrics. We then show the benefits of HCM: we first use it as a framework for identifying the structure and evolution of contributor interactions, and then show that it can help us predict the contributor roles (Section V).

Scope: This work is motivated by both practical software engineering concerns and the need to model software development as an evolving complex system. First, managers may want to assess developer contribution in objective ways, for, say, rewarding key people; the counterpart in the open-source world is "promoting" developers based on merit, as meritocracy is a fundamental tenet (cf. prior Example 2). Second, software managers can use our approach to identify weaknesses in the development process, e.g., single nodes of failure, imbalance in the flow of development. Third, the HCM level (a simple 1–4 number) acts as a proxy for identifying where a developer stands in the hierarchy; this information is useful to both researchers in general and contributors within that project, though Mozilla and Eclipse have declined to provide us with the access level for their contributors due to privacy concerns [7], [22]. Finally, we tackle a long-standing challenge in software maintenance: detecting intrinsic emerging patterns in large, long-term projects [23]–[25].

II. Data Collection and Processing

A. Data Collection

We used data from the Eclipse and Firefox projects: source code, patches, change logs, and bug reports. We analyzed their histories from inception (2001 for Eclipse, 1998 for Firefox) up to April 2010. We use source code information, available in the version control system, to construct source code-based expertise profiles. For each source file, we extract its contributors along with the timestamps of their contributions, and diffs (patches) for each commit. For source code, we analyzed Eclipse versions 1.0 to 3.6.1; for Firefox we analyzed versions 0.8 to 3.6. We use bug report information, available in the bug tracker, to construct bugfix-based expertise profiles. For each bug report, we extract the sequence of assignees and comments. We considered Eclipse Platform bug numbers 1 to 306,296 and Firefox bug numbers 37 to 549,999 (from Mozilla's Bugzilla bug tracker). In Table I, we present a quick overview of data collection and usage.

TABLE I. Data collection sources and uses.

Source	**Raw Data**	**Expertise Profile** (Sec. II-B)	**Role Profile** (Sec. III)	**HCM** (Sec. IV)
Bug tracker	Contributor ID	✓	✓	✓
	Timestamp	✓	✓	✓
(Bug	Severity	✓		
report,	Task: triaged		✓	
Bug	Task: tested		✓	
activity)	Task: assisted		✓	
	Task: analyzed		✓	
	Type (bug/enhancement)	✓		
Source	Committer ID	✓	✓	✓
Code	Log message		✓	
Repository	LOC added	✓		
	Timestamp	✓	✓	✓
(Commit	Files changed	✓	✓	✓
logs)	Bug/Enhancement ID		✓	

B. Expertise Profiles and Metrics

In this section, we introduce **expertise profiles** and provide details on the process and metrics we used to construct these profiles for each contributor. For each member D of a project, we define two kinds of expertise profiles: a *bug-fixing profile*, and a *source code profile*. The rationale for using two profiles is to capture the two major ways of contributing—bug-fixing or development—especially in open source projects. Since many open source projects use separate systems for version control and bug tracking, we performed entity resolution to determine when version control id C_D and bug tracker id $B_{D'}$ correspond in fact to the same individual, i.e., $D = D'$.

We define the **bug-fixing expertise profile** of a contributor D as a tuple ($bugcount_D$, $bugsev_D$, $bugseniority_D$, $bugsfixed_D$) where $bugcount_D$ is the total number of bugs D has been associated with, i.e., D was assigned to fix at some point in time; $bugsev_D$ is the average severity score of all bugs D has fixed;[3] $bugseniority_D$ represents the first and last times D has fixed a bug, as recorded in the bug tracker, and $bugsfixed_D$ is the percentage of bugs D could fix, relative to the total number of bugs assigned to D.

We define the **source-code expertise profile** of a contributor D as a tuple: ($codelines_D$, $files_D$, $codeseniority_D$, $ownership_D$) where the contents are defined as follows: $codelines_D$ is the number of lines of code D has committed—we identify all the patches D has submitted, and from each patch we extract the number of source code lines added or changed. The rationale for using this metric is that the more source code D has contributed, the higher D's expertise

[2]Our discussions with senior developers and managers from the Mozilla and Eclipse projects [21] indicate that operationalizing and determining roles is an important problem in open source project management.

[3]Firefox and Eclipse use a 1-to-7 scale for bug severity (1=Enhancement, 2=Trivial, 3=Minor, 4=Normal, 5=Major, 6=Critical, 7=Blocker).

level is; $files_D$ is the number of files D has worked on. The rationale for using this metric is that the more files D has worked on, the higher D's expertise level is. We define $codeseniority_D$ as the time difference D's first and last commits, as recorded in the project's version control system. Note that if D has only committed once, in our definition D's seniority is zero. A contributor to a software module is someone who has made commits to the module. To quantify the level of involvement between a contributor D and a module C, we define *ownership ratio* as the ratio $R_{DC} = \frac{LOCcommitted_D(C)}{LOCcommitted_{total}(C)}$, i.e., the percentage of lines of code committed by D to C relative to the total number of lines of code committed to C. Note that our definition of ownership is different from Bird et al.'s [13] (which uses the proportion of number of commits) because in the projects we studied we found very low correlation, 0.1403, between the number of commits and the lines of code committed. For each module, based on the ownership ratio R_{DC} we define D's ownership profile (owner, major or minor contributor) in the following table; the first line indicates the ratio, the second line indicates the profile.

$R_{DC} < 5\%$	$5\% \leq R_{DC} < Highest$	$Highest$
Minor contributor	Major contributor	Owner

We have chosen the 5% cut-off based on the cumulative distribution function (CDF) observed in our projects.

C. Feature Selection

We now present statistical evidence that our expertise attribute selection is precise (i.e., all constituent attributes of expertise profiles defined in Section II-B are relevant and there are no redundant attributes). Correlation-based feature selection is a standard machine learning method for filtering out features (attributes) which have strong correlation between them, thus retaining only those features that are independent [26]. We ran a Pearson's correlation test between all pairs of source-code based and bug-fix based expertise attributes, including ownership (Section II-B), and report the results in Tables II and III respectively. We found low pairwise correlation between most features at a statistically significant p-value of 0.01, hence we conclude that, for the projects we considered, feature selection is precise.

D. Contributor Distribution

We now proceed to showing how the breadth and depth of expertise profiles form a firm basis for conducting empirical studies on developer expertise. In particular, we focus on answering two questions.

When do contributors join a project, and for how long do they stay associated with a project? Figure 1 shows the distributions for bugfix- and source code-based seniorities in our examined projects. In each graph, the x-axis shows seniority, in years, and the y-axis shows the number of contributors that have that seniority. Figures 1(a) and 1(b) show bugfix-induced seniority distributions in Eclipse and Firefox. Eclipse Platform has had 8,856 bug fixers over its lifetime; 80.79% of those have seniority less than one year. Firefox has had 19,286 bug fixers over its lifetime; 75.18% of those have seniority less than one year. Note how these values reveal high turnover rate in our examined projects. Figures 1(c) and 1(d) show source code-based seniority distributions in Eclipse and Firefox. Eclipse has had 210 contributors over its lifetime; 58.57% of those have seniority less than one year; Firefox has had 519 contributors over its lifetime; 58.54% of those have seniority less than one year.

What is the contribution distribution in large projects? To ensure that data sets used for our analysis are representative of collaborative software projects, we conducted a contribution distribution analysis. We report three relevant observations about these distributions for Firefox and Eclipse. First, on average, 16.26% (Firefox) and 21.91% (Eclipse) of contributors work on the same file. Second, a contributor on average works on 2.58% (Firefox) and 3.71% (Eclipse) of all project files during their tenure. Third, 27.56% and 18.15% contributors for Firefox and Eclipse respectively have been *owners* of at least one file; and 16.72% and 7.32% contributors for Firefox and Eclipse respectively have been *authors* of at least one file. These numbers demonstrate that Firefox and Eclipse are large projects with a wide range of contribution in terms of major–minor contribution and authorship. Similarly, Figure 1 shows that there is significant variation in seniority which helps us understand the effects of seniority on quality of contribution.

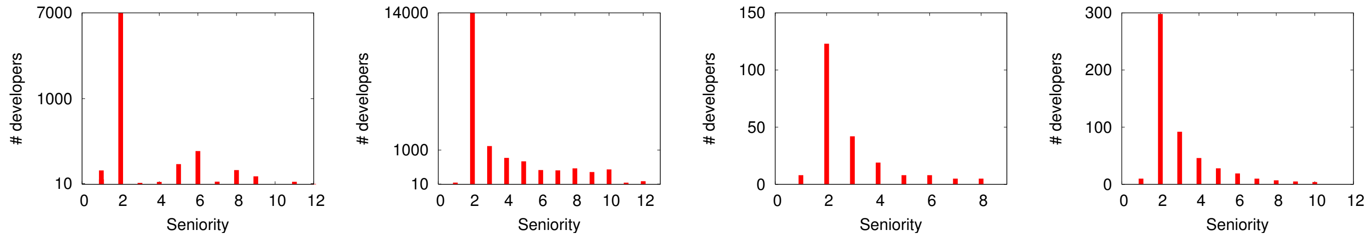
In this section, we have defined a range of expertise metrics and showed that they are not correlated. While metrics indicate contribution *quantity*, they are not a good indicator of the *nature* of contributions; therefore, we will proceed to define seven different types of contribution called *roles*, and show how a hierarchy model is an effective predictor of roles.

III. Contributor Roles

In software projects, an individual's contributions involve more than adding code or fixing bugs. In this section, we operationalize seven roles that capture several aspects of software engineering. Although we do not claim that they capture all facets of open source software engineering, we argue that they are a good start towards a systematic framework, that we develop here. Role operationalization is based on the nature and timing of developer contributions. For now, we do not assign any threshold to the frequency of contribution to mark a contributor as an active participant for a specific role; in the future, we intend to vary the threshold and evaluate its effect on our analysis.[4] The roles are not mutually exclusive, and a contributor can serve multiple roles during her association with a project. Next, we define each of these roles that form the bug- and source-code based profiles.

Triagers: Contributors who triage bugs are indispensable in large projects that receive hundreds of new bug reports every day [27]. A triager's role ranges from marking duplicate bugs, to assigning bugs to potential contributors, to ensuring that re-opened bugs are re-assigned, and to closing bugs after they have been resolved. *In our case, by triaging we refer to contributors who inspect the bug report and identify potential bug fixers (i.e., other contributors who could fix the bug).* As shown in our previous work [27], choosing the right bug fixer is both important and non-trivial: when a bug assignee cannot fix the bug, the bug is "tossed" (re-assigned), which prolongs the bug-fixing process. Jeong et al. [28] introduced the concept of *tossing graphs*, where nodes represent developers and edges represent bugs being tossed; a bug's lifetime (assignment, tossing, and fixing) can be reconstructed from its corresponding

[4]We list this as a potential threat to validity in Section VI.



(a) Eclipse (bug-based) (b) Firefox (bug-based) (c) Eclipse (source-based) (d) Firefox (source-based)

Fig. 1. Bugfix-induced and source code-induced seniority.

TABLE II. CORRELATION BETWEEN BUG-FIXED PROFILE ATTRIBUTES (P-VALUE ≤ 0.01 IN ALL CASES).

Pairwise correlation values								
	$bugseniority$		$bugsev$		$bugcount$		$bugsfixed$	
	Eclipse	Firefox	Eclipse	Firefox	Eclipse	Firefox	Eclipse	Firefox
$bugseniority$	1		0.2728	0.2540	0.1481	0.2531	0.0068	-0.0362
$bugsev$			1		0.3731	0.4264	0.1137	0.0182
$bugcount$					1		0.0028	0.0604

TABLE III. CORRELATION BETWEEN SOURCE-CODE PROFILE ATTRIBUTES (P-VALUE ≤ 0.01 IN ALL CASES).

Pairwise correlation values												
	$codeseniority$		$codelines$		$files$		$owner$		$major$		$minor$	
	Eclipse	Firefox	Eclipse	Firefox	Eclipse	Firefox	Eclipse	Firefox	Eclipse	Firefox	Eclipse	Firefox
$codeseniority$	1		0.4722	0.3637	0.2604	0.3119	0.3114	0.0138	0.1434	0.0525	0.0107	0.1155
$codelines$			1		0.4231	0.7063	0.1781	0.4578	0.0248	0.0978	0.0034	0.0218
$files$					1		0.1620	0.1751	0.0766	0.0439	0.1178	0.2796
$owner$							1		0.0593	0.1584	0.1396	0.0037
$major$									1		0.0446	0.0895

tossing path in the tossing graph. We identify a triager as the first contributor in the tossing path to "assign" a bug to another contributor. Note that bugs with assignee status "Nobody's OK to work on it" might have contributors assigning themselves as the bug-fixer. We do not consider self-assignment as triaging.

Bug analysts: Contributors who help in analyzing the bugs play an important role in the bug-fix process. *In our study we label contributors as bug analysts if they perform one of the following decision-making tasks*: (1) prioritizing bugs, (2) deciding when to assign "won't fix" status to bugs, (3) labeling a feature enhancement for future release, (4) identifying duplicate and invalid bugs, and (5) confirming a new bug as a valid bug by reproducing the errors as per the description in the bug report. To find bug analysts, we text-mined the activity page of each bug to find the list of developers who have performed one or more of the tasks (1)–(5) described above.

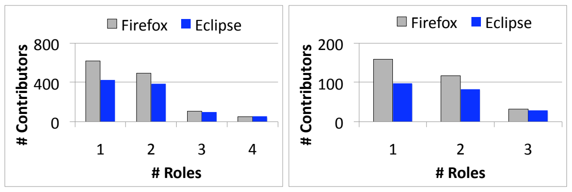
Assists: *Contributors who have found the right person to fix a bug at least once are defined as assists in our model.* This role captures the "*who knows who knows what*" relationship in a social network. In our case, an assist is the second-to-last person in the bug tossing path, i.e., D is an assist if D assigned the bug to a contributor E (after it was tossed among other contributors) and E finally fixed the bug. Note that, to identify assists, we only consider bugs whose the tossing path length is greater than 1. We identify assists automatically by extracting the second-to-last contributor on each tossing path.

Patch tester: After a patch (potential fix) for a bug has been submitted, certain contributors test the patch, to ensure that the bug has been correctly resolved. We identify these contributors, i.e., patch testers, from bug report comment sections where they report their results from testing the newly-submitted patch.[5] Additionally, a tester may also suggest ways of improving the code quality. Patch testers are mined from the activity page (where an individual is explicitly assigned as the patch-reviewer) and the comment section of a bug report (where they suggest appropriate changes for improvement). *In our model, a patch tester is a developer who has reported patch testing results at least once.*

Patch-quality improvers: These contributors improve the quality of bug-fixing patches submitted by other contributors using either manual inspection or automatic code review tools (e.g., Mozilla uses JST Review Simulacrum) to maintain a level of consistency in design and implementation practices as set by project managers.[6] Improvements involve adding documentation, cleaning up the code, ensuring compliance with coding standards, etc. We label a contributor D as a patch-quality improver if we find that D has modified newly-submitted patches and the log message contained strings such as "cleaned patch", "added documentation", "simplified string definition", "changed incorrect use of variable", etc. Identifying such messages automatically for large source-code repositories like Firefox and Eclipse is a non-trivial task. We used a text mining technique to extract such messages: (1) identify all log messages that are submitted with the same bug-ID for the same file, (2) sort the log messages chronologically, and (3) check if the log messages submitted after the initial patch contain words like "added", "changed", "simplified", "removed", "reverted", "replaced", "rewrote", "updated", "renamed", etc. *In our model, a patch-quality improver is a developer whose log messages contain the aforementioned keywords at least once.*

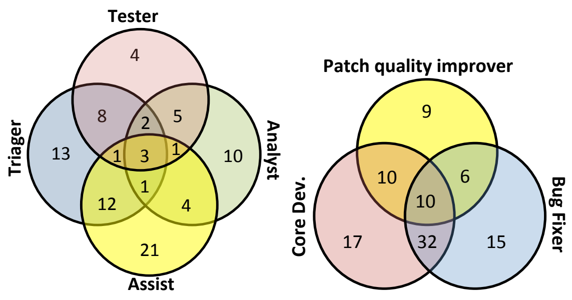
[5] For example, Mozilla bug 50212 (https://bugzilla.mozilla.org/show_bug.cgi?id=50212) shows how a contributor X plays the role of tester for contributor Y (comment 4).

[6] Mozilla coding guidelines:https://developer.mozilla.org/en-US/docs/Developer_Guide/Coding_Style; Eclipse coding guidelines:http://wiki.eclipse.org/Development_Conventions_and_Guidelines.



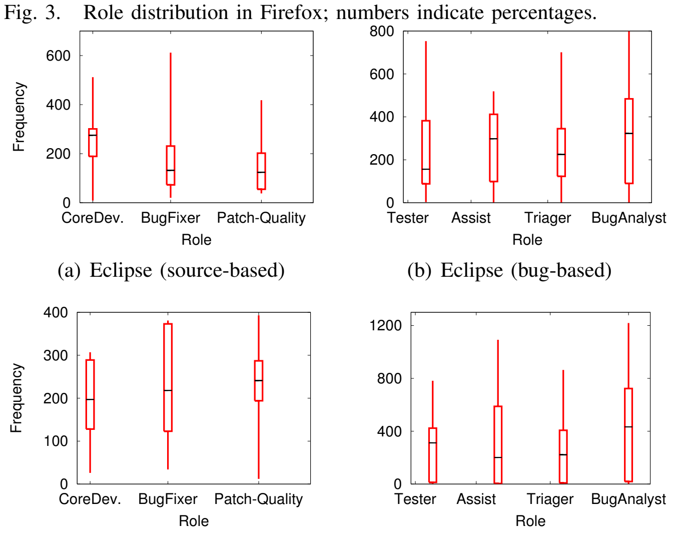
(a) Bug-based roles (b) Source-based roles

Fig. 2. Role frequency.



(a) Bug-based roles (b) Source-based roles

Fig. 3. Role distribution in Firefox; numbers indicate percentages.



(a) Eclipse (source-based) (b) Eclipse (bug-based)

(c) Firefox (source-based) (d) Firefox (bug-based)

Fig. 4. Frequency of contribution for each role.

***Core developers**: We define as core developer a contributor who has added new code to the source code repository in response to a feature enhancement request or has added code that does not correspond to a bug-fix.* This way we ensure separation between contributors who perform adaptive/perfective maintenance and those who perform corrective maintenance (bug-fixers, described below). A contributor is labeled as a core developer if the log message of the code churn he has committed contains a feature enhancement ID.

***Bug fixers**: We tag a contributor D as a bug-fixer if D has added code for fixing a bug.* In other words, a bug fixer performs corrective maintenance. A commit is identified as corrective maintenance by cross-referencing the bug ID associated with the log message with the bug type (i.e., defect or enhancement) in the bug database. A contributor is labeled as a bug fixer if the log message of the code churn he has committed contains a bug ID.

Analysis of role distributions: To illustrate how contributors serve multiple roles in a project, we provide three analyses. In Figure 2 we show the absolute number of contributors (y-axis) who have served one or more roles (x-axis); as expected, the bulk of contributors have only served one role, with much smaller numbers serving all 4 bug-based roles or all 3 source-based roles. In Figure 3 we show the distribution and overlap of roles, in percentages, for Firefox[7] we now proceed to explain the graph. For example, among those developers who have ever served source-based roles: 17% have only served as core developers, 32% have served as both core developers and bug fixers, and 10% have served all three roles. In Figure 4 we characterize the frequency distributions for each role, across all contributors for that role. Each candlestick represents the minimum, first quartile, second quartile, third quartile and maximum; the black bar is the median. For example, for core developers in Eclipse (leftmost candlestick in Figure 4(a)): the minimum number of role servings was 9, the median number was 275, and the maximum number was 512.

Role profile: We define two kinds of role profiles of a contributor D: bug-based role profile and source code-based role profile.

Bug-based role profile is a tuple $\langle Triager, BugAnalyst, Assist, PatchTester\rangle$ and source code-based profile is a tuple $\langle CoreDeveloper, BugFixer, PatchQualityImprover\rangle$. Each metric can have a integer value; if the value is zero, it indicates that the contributor have never served the role or else any non-zero value would indicate number of times she has served the role (frequency of contribution). For example, a bug-based role profile (D) = $\langle 2, 0, 3, 5\rangle$ implies that contributor D has served the role of triager twice, never served the role of bug analyst, served the role of assist 3 times and the role of patch tester 5 times.

IV. HCM: Our Graph-Based Model

With the contributor role definitions in hand, we now proceed to defining a *hierarchical contributor model* (HCM) that emerges from the collaboration among contributors in the course of source- and bug-based collaboration. The model has several key advantages: (a) it captures the hierarchy and "importance" of contributors, in a way that was hard or impossible to do with conventional expertise metrics, (b) we show how, by using our model, we can in fact infer roles and contributions with relatively high accuracy using raw data: source-code repository and bug database, without the need of the role profiles, which is not always available for all software projects, (c) it captures the stability of developer interaction. The model is based on two collaboration graphs, which we describe next.

Bug-based collaboration graph: $\mathbb{G}_{Bug} = (V, E)$ where V, the set of vertices, represents contributors involved in fixing bugs, and an edge $e(u, v)$ indicates that contributors u and v have worked on the same bug, for at least one bug. A contributor u is involved in fixing a bug if he was assigned the bug, or changed the bug severity or priority, or added information about the bug, or has tested a patch. This information is available from the activity page of each bug in Bugzilla.[8]

Source-code based collaboration graph: $\mathbb{G}_{Source} = (V, E)$ where V, the set of vertices, represents contributors who have edited a source-code file, and an edge e between two vertices

[7]Eclipse role distribution percentages are similar; we omit them for brevity.

[8]Bug-based collaboration graphs are not the same as bug-tossing graph, which capture *who passed a bug to whom* [28].

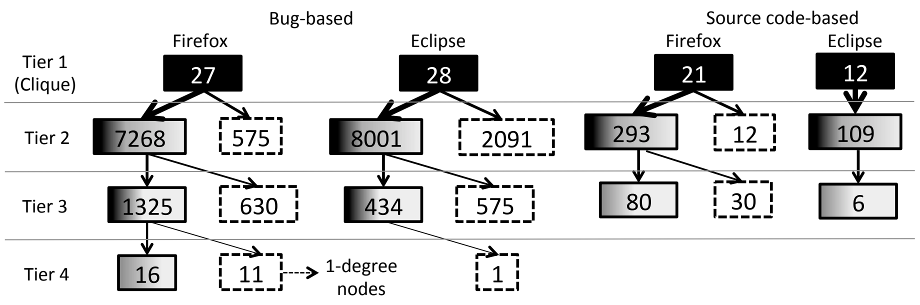


Fig. 5. HCM (hierarchy and tier distributions) for bug-based and source-based contributor collaborations.

u and v implies that they have worked on at least one file. The file modification can be: (a) addition of code for a new feature or (b) fixing a bug in that file. Note that, as defined above, the graphs are undirected, and this is how we use them in this section. In section V-A1, where we analyze the graphs' topology, we add edge directions.

The emerging structure: hierarchy and tiers. We want to identify structure in our two types of collaboration graphs. The graph mining literature offers many ways to analyze graphs structure, e.g., in terms of node clusters or node importance. Given our interest in identifying "importance" of contributors, we use the following insight: important contributors are likely collaborating with many other contributors. With this in mind, we follow the process below, which was also used successfully in a different context [29].

1. Determining the "center." Intuitively, we want to identify the *largest* clique with the highest-degree nodes in the graph. The process starts from the highest-degree node, and includes all nodes that are connected with all the other nodes in the clique. We refer to the nodes in the clique as *Tier 1*, as shown in Figure 5.

2. Determining tiers recursively. Given the definition of tier 1, we define subsequent tiers using connectivity to the previous tier. Specifically, we use the following recursive procedure: a node belongs in tier k if and only if it is connected with at least one node in tier $k-1$.

3. Distinguishing one-degree nodes. In an additional step, we add more information to our model by reporting one-degree nodes (nodes with only one edge) within each tier.

Compact and informative representation. The above process leads to a compact representation of our graphs; Figure 5 shows the HCMs for bug-based and source-code-based collaboration graphs. To convey more information, we introduce two features. First, we use darker shades to indicate tighter connectivity internally within each box: black signifies the clique, while the one-degree hanging nodes to the right are white, which represents no connectivity. Second, the width of the edges between two tiers is proportional to the number of edges across these tiers.

Why is the HCM useful? The advantage of the HCM lies in its simplicity and the amount of information it can "encode": intuitively, it maximizes the ratio of information over model complexity. The HCM level (a simple 1–4 number) is an effective indicator of where a developer stands in the hierarchy—this information is useful to both project insiders and outsider, e.g., researchers. Some observations emerge from examining the model: (a) there exists a clique of non-trivial size (12–27 or 0.26%–9.44% of the nodes), (b) there are relatively few tiers, between 3 and 4 in our graphs, even though the graphs have more than 10,000 nodes, (c) there is a non-trivial number of one-degree nodes (12%, 24%, and 10%, for the graphs, except a really small one for Eclipse source-code), and (d) many one-degree nodes connect to the tier 1 nodes.[9]

The model structure is "aligned" with contributor expertise and contribution. High-performing contributors tend to be in lower (numerically) tiers and thus higher in the hierarchy. In Figure 6, we present the results of contributor expertise distribution across the various tiers formed for selected metrics. We find that tier 1 nodes (the clique) have high-values of expertise attributes and the values decrease as we move on from tier i to tier $i+1$. In other words, tier 1 contributors are among the most active and experienced contributors for a project. We argue that the tier of a contributor is a good estimate of his expertise and contributions: a member of the clique is likely a senior contributor, who has fixed high-severity bugs, many bug types, and owns many files; conversely, we expect a contributor from tiers 3–4 to be a junior contributor with low expertise. We substantiate this claim in Section V.

An interesting observation is that nodes in tier 2 are strongly connected with nodes in tier 1. For example, in Firefox, 68.79% of tier 2 nodes connect to 74.07% of the clique (tier 1) nodes. We believe that this strong connectivity indicates two collaboration traits: (1) tier 2 contributors work very closely with senior contributors for maintaining the project, and (2) tier 2 contributors are stable[10] members of the project.

Validity. We manually inspected the profile of each clique member to validate the legitimacy of their presence in tier 1. For Mozilla, we used the super-review status as cross-check.[11] We found that, out of 28 super-reviewers, 21 are in the source-code based clique and 24 are in the bug-based clique. The remaining super-reviewers who are not in the clique belong to the second tier and have high in- and out-degrees. For Eclipse we cross-checked the clique against the list of Foundation Council members.[12] We found that 23 out of 54 council members are in the bug-based clique while all 12 members of the source-code clique are part of the council.

[9]We found that our graphs' assortativity is negative or around zero (between -0.3 and 0.011) which contradicts a natural inclination to assume that high degree nodes are in higher tiers.

[10]It is important for managers in open source projects to identify which project members are stable. Our techniques can help managers identify these stable contributors easily, as they are nodes with high in- and out-degrees.

[11]The super-reviewers group is a set of senior, experienced contributors who can add value across the codebase in some specific ways separate from domain expertise [30]. At the time this work was carried out, only 28 contributors were assigned super-review status.

[12]Eclipse Foundation Council, http://eclipse.org/org/foundation/council.php

Disconnected nodes. We use connectivity to establish our model, and we find that more than 95% of the nodes form a large connected component, and thus represented in our model. The remaining nodes (<5%) are disconnected from this connected component, and form mini-graph structures with 2–13 nodes each. We found that all these nodes have seniority one year, which is the lowest, and their expertise profiles are low (e.g., $bugseniority \leq 2$, $bugcount \leq 7$, $bugsev \leq 1.44$).

V. Using The HCM Model

In this section, we show how HCM can help us conduct studies that reveal interesting properties in terms of structure and evolution of the collaboration relationships. We also show how we can use the concise information encoded in the model to predict the roles of contributors.

A. Collaboration and HCM Evolution

1) Intensity of Collaboration: We refine our HCM by moving to directed graphs with weighted edges: the edge weight is the number of times two contributors have interacted for bug-fix or code-changes. Intuitively, this weight represents the intensity of the collaboration between contributors: if contributor v acted on a file (or bug) before contributor w did, then their common edge is directed, $v \rightarrow w$; if there is a common file that contributor w acted on first, then the reverse edge $w \rightarrow v$ is also represented in the graph. We analyze how strongly the graph is connected considering collaboration intensity. We define a weight threshold t_{cut}, which we use to filter out all the edges with weight $w \leq t_{cut}$. When setting $t_{cut} = 1$, we found that 72.77% nodes in Firefox and 64.93% of nodes in Eclipse become disconnected from the initial HCM graph. We then increased our threshold $t_{cut} = 2, 3, \ldots$ and observed an interesting phenomenon. The original connected component shrinks significantly if we remove edges with $w < 3$ for Firefox and $w < 5$ for Eclipse, but after that, even if we increase t_{cut}, (until $t_{cut} = 118$ for Eclipse and $t_{cut} = 206$), the connected component do not change. We find that the connected component consists of contributors only from tier 1 and tier 2, which are connected with high-weight edges. This shows that *the majority of collaborations take place between the top two layers of the network model*. This agrees with our results in section IV, where we found that apart from fixing bugs, contributors from these layers serve multiple non-technical roles.

2) Evolution of collaboration graphs: To understand how the collaboration graphs evolve over time, we built three snapshots for years 2006, 2008, 2010 for both Firefox and Eclipse.[13] We found that between 2006–2010, the size of the graphs doubled for Firefox and tripled for Eclipse. By further studying the HCM model of each instance, we find three interesting aspects:

The clique grew significantly: In Firefox, the clique grew from 4 to 11 to 27 contributors. In Eclipse, the clique size grew from 9 to 16 to 28. Note that this growth rate is much higher than the theoretical growth rate of a clique in a scale-free network ($\log \log N$ with the size of the network [31]).

[13]Firefox and Eclipse had their first official releases in 2004, and the number of contributors has increased steadily since then. We chose 2006 as starting point to ensure the samples are sizable and representative of contributors from all components.

The clique is stable over time: We observed that only 3 contributors in Firefox and 1 contributor in Eclipse were discarded from the clique (tier 1), i.e., were present in the 2006 snapshot but not anymore in the 2008 and 2010 snapshots. This indicates the stability of the clique and strengthens our claim that contributors in the clique serve all possible roles. If we found that contributors in the clique are unstable, it would have reduced the confidence level of our role-prediction accuracy.

Climbing up in the hierarchy requires work: We find that contributors who advance to an upper tier show a significant increase in their expertise profile metrics (number of bugs fixed, eLOC added, etc.) from the previous snapshot of the graph. This observation validates our claim that the tier a contributor belongs to is an indicator of her expertise level and that promotion from a lower-level tier to higher-level tier would require demonstration of significant contributions. Additionally, this demonstrates that the promotion of a contributor in the expertise hierarchy is *merit-based*. In the future, we plan to study how and when this promotion or tier change occurs, and factors that determine the threshold of this promotion.

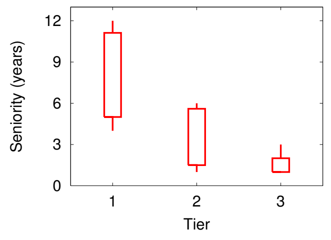
3) Expertise Breadth vs. Depth: We analyze how expertise breadth (i.e., familiarity with *multiple* components in a large project) is different from depth (i.e., familiarity with a *single* component). We hypothesize that contributors who gain familiarity with multiple components of the same project gain expertise quicker than those contributors familiar with a single component. To validate this hypothesis, we update each contributor's profile with a list of components they have worked on in Firefox and Eclipse. We found that contributors in the clique have worked on at least 80.71% (Firefox) and 69.88% (Eclipse) of sub-components. Also, within the clique we see two different distributions for both projects: (1) breadth-experts: contributors who have worked on at least 52.31% (Firefox) and 44.55% (Eclipse) of the components, and (2) depth-experts: the remaining contributors who have worked on a single component only. We found that people who are breadth-experts are senior contributors, as opposed to depth-experts who are junior members. This indicates that when contributors join a project, they start gaining expertise in a single component; as their expertise grows over time, their familiarity (and therefore breadth-expertise) broadens.

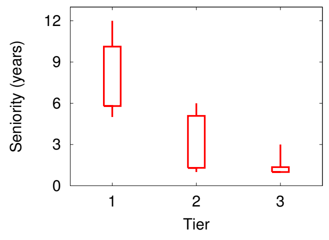
These findings confirm the utility of our approach in determining whether these projects are indeed meritocracies as stipulated in their charters.

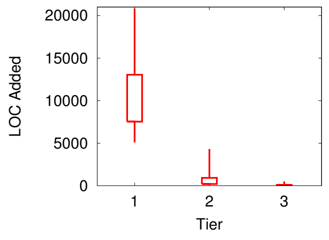
B. Predicting Role Profiles Using HCM

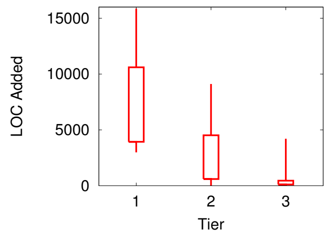
The HCM encodes significant information concisely, and an indication of this is that it can be used to predict the role profile of a developer D. This ability to predict role profiles from the HCM is crucial, since, as we explained in Section IV, the HCM can be constructed even for projects where certain source code and bug information might be unavailable. Figure 7 shows an overview of the process used for constructing and validating our predictor model—note how constructing HCM requires only a subset of bug and source data. We now proceed to defining our model, then evaluating its accuracy, and finally showing that predictors constructed using standard expertise metrics have poor prediction accuracy.

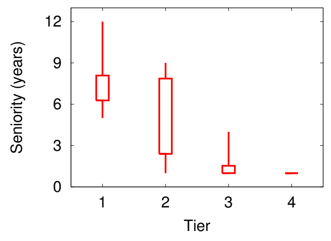
Defining the prediction model. We refine the initial graphs to be directed (Section V-A1) and thus to each node

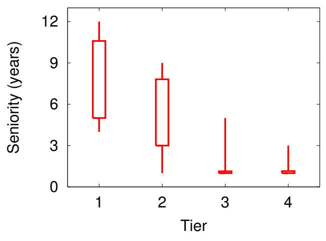

(a) Eclipse (source-based)

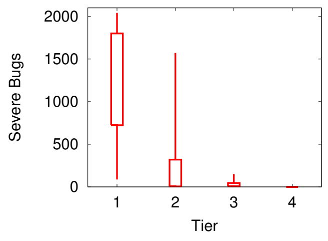

(b) Firefox (source-based)


(c) Eclipse (source-based)


(d) Firefox (source-based)


(e) Eclipse (bug based)


(f) Firefox (bug based)


(g) Eclipse (bug based)

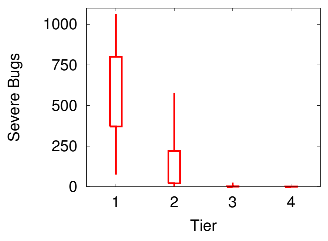

(h) Firefox (bug based)

Fig. 6. Tier distribution range per expertise metric.

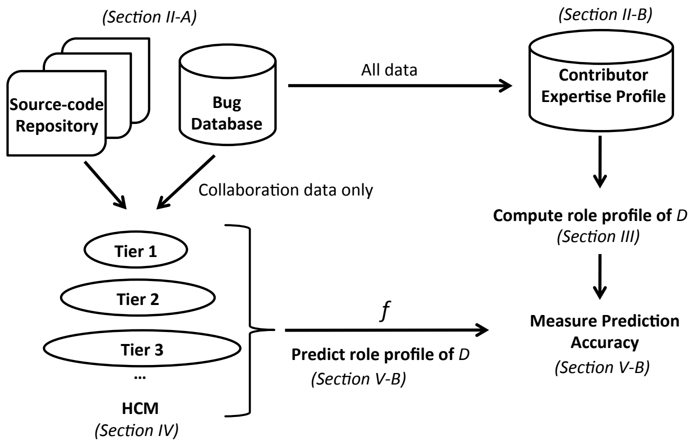


Fig. 7. Measuring prediction accuracy.

Algorithm 1 Definition of f for predicting bug-based role profile of contributor D

```
Input: $Tier_D$, $InDegree_D$, $OutDegree_D$
Output: $RoleProfile_D$
Description:
  if $Tier_D = 1$ then
    $D$ has served ALL Roles
  else if $Tier_D$=2 then
    if $InDegree_D \geq 80\%$ & $OutDegree_D \geq 80\%$ then
      $D$ has served as an Assist and Triager
    if $InDegree_D \geq 80\%$ & $OutDegree_D < 80\%$ then
      $D$ has served as a Patch Tester
    if $InDegree_D < 80\%$ & $OutDegree_D \geq 80\%$ then
      $D$ has served as an Assist
    if $InDegree_D < 80\%$ & $OutDegree_D < 80\%$ then
      $D$ has served as a Bug analyst
  else if $Tier_D \geq 3$ then
    $D$ has served NO Roles
```

(developer) D, in addition to level $Tier_D$ we associate in- and out-degrees, $InDegree_D$ and $OutDegree_D$. With that in hand, we construct a role predictor based on the HCM, i.e., a function f that, given HCM data for developer D, outputs the role profile of D:

$$< RoleProfile_D >= f(Tier_D, InDegree_D, OutDegree_D)$$

The definition of function f for bug-based roles is shown in Algorithm 1, while for source-based roles in Algorithm 2; we now proceed to explain these definitions. The cases $Tier_D = 1$ are based on the observation that contributors in the upper hierarchy (clique) are likely to have participated in all roles. Similarly, $Tier_D \geq 3$ indicates that contributors at the bottom of the hierarchy are not likely to have participated in any role.

We derived the middle cases, i.e., $Tier_D = 2$, based on analyzing the CDF of the in- and out-degrees and choosing an 80 percentile "high" threshold[14] for $InDegree_D$ and $OutDegree_D$. We observed that both Assists and Triagers accept bugs from others (from the previous assignee in the case of Assists, and from bug reporters in case of Triagers) therefore these contributors have high $InDegree_D$; they also assign or pass bugs to other contributors, therefore they have high $OutDegree_D$. Hence, in Algorithm 1, **if** $InDegree_D \geq 80\%$ & $OutDegree_D \geq 80\%$ then D has served as an Assist and Triager . Patch Testers are assigned lots of patches to test (high $InDegree_D$) but do not usually re-assign patches (low $OutDegree_D$). The other cases follow by a similar argument.

For Algorithm 2, note that Core developers and Fixers are assigned new features or bug fixes by others (therefore high $InDegree_D$) and then they pass on their code to either Patch-quality improvers or Testers (hence high $OutDegree_D$) Patch-quality improvers accept code improvement requests from others (high $InDegree_D$) but do not re-assign code (low $OutDegree_D$). The other cases follow similarly.

We now evaluate the effectiveness of the HCM-based predictor model. As shown in Figure 7, we use the function f defined above to predict the role profile of D, and then compare this predicted role profile with the role profiles we computed in Section III (i.e., the latter serves as reference output). Specifically, prediction accuracy is the ratio of two numbers: the number of developers we correctly predict have served role R over the total number of contributors that we have identified to have role R. We report prediction accuracy in columns 2 and 3 of Table IV. We found that the highest prediction accuracy (75.98%) was achieved when predicting Assists in Firefox. The lowest prediction accuracy (47.22%)

[14]The 80 percentile cutoff might vary across projects (Section VI).

Algorithm 2 Definition of f for predicting source-based role profile of contributor D

```
Input: $Tier_D$, $InDegree_D$, $OutDegree_D$
Output: $RoleProfile_D$
Description:
  if $Tier_D$ = 1 then
    $D$ has served ALL Roles
  else if $Tier_D$=2 then
    if $InDegree_D \geq 80\%$ & $OutDegree_D \geq 80\%$ then
      $D$ has served as Core developer and Bug fixer
    if $InDegree_D \geq 80\%$ & $OutDegree_D < 80\%$ then
      $D$ has served as a Patch-quality improver
    if $InDegree_D < 80\%$ & $OutDegree_D \geq 80\%$ then
      $D$ has served as Bug fixer
    if $InDegree_D < 80\%$ & $OutDegree_D < 80\%$ then
      $D$ has served NO Roles
  else if $Tier_D \geq 3$ then
    $D$ has served NO Roles
```

TABLE IV. ROLE PROFILE PREDICTION ACCURACY USING HCM.

Role	**Prediction accuracy (%)**	
	Eclipse	Firefox
Patch tester	69.62	66.28
Assist	67.73	75.98
Triager	59.06	60.37
Bug analyst	53.18	69.45
Core developer	70.96	62.89
Bug fixer	65.71	58.25
Patch-quality improver	61.80	47.22

was attained when predicting Patch-quality improvers in Firefox. Note that, even though 47.22% seems low, it is by no means comparable to coin-tossing: per Figure 3, serving or not serving a role are not equally probable.

Clustering contributors: We also investigated whether expertise metric values can be used to predict roles: can we form clusters based on expertise metric values that would correspond to roles? To answer this question, we first used the contributor expertise profiles (the tuples described in Section II-B) as input to the EM clustering algorithm [32].[15] After EM has determined clusters, we measured the fit between EM clusters and roles as the ratio between the number of pairs of contributors D_1, D_2 who serve role R and are in the same cluster over the total number of pairs of contributors D_1, D_2 who serve role R. Put another way, this ratio tells us how many developers D with similar role R are within a cluster. For brevity we omit details, but we found the fit to be low (minimum 6.36%, median 15.62%, maximum 30.92%) for all roles in both Firefox and Eclipse. These findings suggest that standard expertise metrics do not make good role indicators.

Discussion: The main point of comparing roles determined via EM cluster with roles determined from HCM is to show the inadequacy of determining roles using the initial graphs or the raw contributor activity data. The fact that HCM can provide more than 50% precision is a good indication that the model captures our intended characteristics.

VI. Threats to validity

External Validity: Our expertise profiles and role definitions assume access to the source and bug repositories; this data might not be available in all projects, hence by selecting projects which have this information—Firefox and Eclipse—our study might be vulnerable to selection bias. We only studied open source projects; commercial projects might have different ways to quantify contributor expertise and roles.

[15]We used the *Akaike Information Criterion* (a standard machine learning metric [33]) to determine the optimal number of clusters, i.e., balance between a good fit and a small number of clusters, to avoid over-fitting.

Internal Validity: Our bugfix-induced data relies on bug reports collected from Bugzilla at the time the paper was written. Future changes in bug status (e.g., if closed bugs is re-opened) or bug severity might affect our results.

Construct Validity: We assume that our metrics actually capture the intended characteristic, e.g., the expertise attributes we use accurately models an individual's expertise. We intentionally used multiple bug-fix induced and source-code based metrics to reduce this threat. The roles we operationalize do not use cut-off points for frequency of contribution; therefore we do not to distinguish between expert and non-expert contributors *within a specific role*. We based our role thresholds on the CCDF of in- and out-degrees which might vary with projects. In the future, we intend to vary the threshold and evaluate its effect on our analysis.

Content Validity: The assignee information in Bugzilla does not contain the domain of contributors' email addresses. Therefore, we could not differentiate between users with the same email username but different domains (in our technique, bugzilla@alice.com and bugzilla@bob.com will be in the same bucket as bugzilla@standard8.plus.com). This might potentially lead to loss of prediction accuracy. Similarly, while extracting contributor id's from log messages, we might miss contributors who submit patches via other committers.

VII. Related Work

Contributor Roles: Teyton et al. propose XTic, an automated apporach towards identifying developer expertise [34]. Expertise is a combination of skills and experience. While Teyton et al. focus on the technical-skills aspect of expertise, we focus on experience, which we define as roles. Our roles abstract specific skills required for the role, e.g., for a tester, we do not differentiate if they are expert in JUnit (a Java unit test framework) or Boost (a C/C++/C# test framework). The "core developer" and "tester" role we define in our work consists of programmers with diverse technical skills. Using our technique, we can differentiate developers with similar skills but performing two different roles in the project.

Yu et al. [8] define core members and associate members in ORAC-DR (14 members) and Mediawiki (56 developers). Alonso et al. [9] quantify developer expertise based on the number of associated files and differentiate between developers and contributors in the Apache project (75 developers, 8 years evolution). Based on keyword clouds they term developers as "generalist" or experts, e.g., "security expert." Our work defines and predicts seven fine-grained, stable roles, uses a wide range of expertise metrics, introduce a hierarchy model and is based on larger data sets.

Developer Expertise: Zhou and Mockus [35] studied developer productivity evolution and found that, when measured in tasks per month productivity grows in the beginning but eventually plateaus. However, after adjusting for task difficulty, their findings indicate that developer productivity continues to increase. Mockus and Herbsleb [10] defined individual expertise in terms of EA (experience atoms), basically the number of commits. They found that new developers start gaining expertise and after a certain period of time their expertise tends to remain constant. Fritz et al. [11], [36] and Schuler et al. [37] define expertise as the knowledge of methods that a developer's code calls. They argued that the more developers reuse (or contribute to) existing code, the more knowledge they have about that code. Rahman et al. [14]

categorized developers into generalized experts and specialists based on components they have committed to while bug-fixing. Minto et al. [12]'s EEL tool can recommend expert developers in emerging teams. They use a recommendation algorithm that ranks developers for a given file, and studied three open source projects: Firefox, Bugzilla and Eclipse. Gousios et al. [38] evaluate developer contributions based on LOC worked on, and events associated with. Dominique et al. [39] build expertise from bug reports vocabularies. Bird et al. [13] studied the effects of code ownership in Windows software, and found that code ownership is an effective indicator of developer's knowledge. However, they do not quantify contributor role or expertise using a wide-range of expertise attributes or use contributor hierarchy as a proxy for developer expertise.

Our work is significantly different in four ways: (1) we couple the source-code and bug-based expertise of contributors while all prior studies used only one when quantifying contributor expertise, (2) we define and differentiate a contributor's role from her expertise, (3) we demonstrate that the collaboration-based hierarchy is an effective way to estimate a contributor's role from her expertise profile, and (4) we differentiate between expertise breadth and expertise depth.

Collaboration Graphs and Hierarchy Detection: A rich body of literature [15]–[20] explores contributor collaboration in the context of social networks analysis (SNA). However, there has been no research in the area of extracting expertise hierarchy using contributor collaboration networks to quantify contributor expertise or role. Hierarchy detection has been widely studied in sociology [40], network sciences [41]–[44], and online social networks [45]–[47]. O'Mahony studied the relationship between participation and leadership positions in non-technical tasks like mailing list management in Debian [48]. In contrast, in our study we do not consider leadership as a role or expertise measure. Several SNA tools quantify and qualify large networks, primarily by describing network features through numerical or visual representation [49], [50]; the focus of our study has however been to use SNA-based metrics to build a contributor role-based taxonomy.

VIII. Conclusions

We have studied two large, long-lived projects, Firefox and Eclipse, to operationalize contributor role and expertise. We show that role and hierarchy information can capture a developer's profile and impact in ways current expertise metrics cannot. We have also explored how a contributor's role, breadth and depth expertise evolve over time. We have found that collaboration can be an effective predictor of individuals' roles; and that as contributors' expertise increases, they tend to serve multiple roles in the project.

Acknowledgments

This research was supported in part by NSF grant CCF-1149632. We thank the anonymous referees for their helpful comments.

References

[1] "Mozilla: Mozilla roles and leadership," 2013, http://www.mozilla.org/about/roles.html.
[2] "Eclipse: Eclipse development process," 2011, http://www.eclipse.org/projects/dev_process/development_process_2011.php.
[3] "Ubuntu governance," 2013, http://www.ubuntu.com/about/about-ubuntu/governance.
[4] "Debian project: Why debian for developers," 2013, http://wiki.debian.org/WhyDebianForDevelopers.
[5] "The apache software foundation: How it works," 2013, http://www.apache.org/foundation/how-it-works.html#meritocracy.
[6] F. P. Brooks, Jr., *The mythical man-month*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.
[7] "Mozilla bug 699815," 2011, https://bugzilla.mozilla.org/show_bug.cgi?id=699815.
[8] L. Yu and S. Ramaswamy, "Mining cvs repositories to understand open-source project developer roles," in *MSR*, 2007, p. 8.
[9] O. Alonso, P. T. Devanbu, and M. Gertz, "Expertise identification and visualization from cvs," in *MSR*, 2008, pp. 125–128.
[10] A. Mockus and J. D. Herbsleb, "Expertise browser: a quantitative approach to identifying expertise," in *ICSE*, 2002, pp. 503–512.
[11] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill, "A degree-of-knowledge model to capture source code familiarity," in *ICSE'10*.
[12] S. Minto and G. C. Murphy, "Recommending emergent teams," in *MSR'07*.
[13] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't Touch My Code! Examining the Effects of Ownership on Software Quality," in *FSE*, 2011, pp. 4–14.
[14] F. Rahman and P. T. Devanbu, "Ownership, experience and defects: a fine-grained study of authorship," in *ICSE*, 2011, pp. 491–500.
[15] C. Bird, "Sociotechnical Coordination and Collaboration in Open Source Software," in *ICSM*, 2011, pp. 568–573.
[16] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *MSR*, 2006, pp. 137–143.
[17] P. C. Rigby and A. E. Hassan, "What can oss mailing lists tell us? a preliminary psychometric text analysis of the apache developer mailing list," in *MSR*, 2007, p. 23.
[18] P. C. Rigby, D. M. German, and M.-A. Storey, "Open source software peer review practices: a case study of the apache server," in *ICSE'08*.
[19] A. Meneely and L. Williams, "Secure open source collaboration: an empirical study of linus' law," in *CCS*, 2009, pp. 453–462.
[20] Q. Hong, S. Kim, S. Cheung, and C. Bird, "Understanding a developer social network and its evolution," in *ICSM*, 2011, pp. 323–332.
[21] G. Markham, Personal communication, 2011.
[22] C. Aniszczyk, Personal communication, 2011.
[23] T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri, "Challenges in software evolution," in *IWPSE'05*.
[24] "Mozilla management," November 2011, https://wiki.mozilla.org/Mozillians#Context.
[25] C. Rossi, "Facebook push: Tech talk," May 2011, http://www.facebook.com/video/video.php?v=10100259101684977.
[26] M. A. Hall, "Correlation-based Feature Subset Selection for Machine Learning," Ph.D. dissertation, University of Waikato, 1999.
[27] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," in *ICSM*, 2010.
[28] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *FSE*, 2009, pp. 111–120.
[29] G. Siganos, L. Tauro, and M. Faloutsos, "Jellyfish: A conceptual model for the internet topology," *Journal of Computer Networks*, vol. 8, no. 3, pp. 339–350, September 2006.
[30] "Mozilla super-reviewing policy," 2011, http://www.mozilla.org/hacking/reviewers.html.
[31] H. Reittu and I. Norros, "On the power law random graph model of the internet," *Perf. Eval. 55(1-2): 3-23*, 2004.
[32] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 1–38, 1977.
[33] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, 1974.
[34] C. Teyton, M. Palyart, J.-R. Falleri, F. Morandat, and X. Blanc, "Automatic extraction of developer expertise," *18th International Conference on Evaluation and Assessment in Software Engineering*, 2014.
[35] M. Zhou and A. Mockus, "Developer fluency: achieving true mastery in software projects," in *FSE '10*, 2010, pp. 137–146.
[36] T. Fritz, G. C. Murphy, and E. Hill, "Does a programmer's activity indicate knowledge of code?" in *ESEC-FSE*, 2007, pp. 341–350.
[37] D. Schuler and T. Zimmermann, "Mining usage expertise from version archives," in *MSR*, 2008, pp. 121–124.
[38] G. Gousios, E. Kalliamvakou, and D. Spinellis, "Measuring developer contribution from software repository data," in *MSR*, 2008, pp. 129–132.
[39] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," in *MSR'09*.
[40] R. Stark, *Sociology*. Thompson Wadsworth, 2007.
[41] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *J. ACM*, vol. 46, September 1999.
[42] A. Clauset, C. Moore, and M. E. J. Newman, "Structural inference of hierarchies in networks," in *ICML*, 2006, pp. 1–13.
[43] G. Siganos, S. L. Tauro, , and M. Faloutsos, "A simple conceptual model for the internet topology," *IEEE Global Internet*, 2001.
[44] R. Albert, H.Jeong, and A. Barabasi, "Diameter of the world wide web," *Nature*, 1999.
[45] M. Gupte, P. Shankar, J. Li, S. Muthukrishnan, and L. Iftode, "Finding hierarchy in directed online social networks," in *WWW'11*.
[46] A. S. Maiya and T. Y. Berger-Wolf, "Inferring the maximum likelihood hierarchy in social networks," in *CSE*, 2009, pp. 245–250.
[47] D. Liben-Nowell and J. Kleinberg, "The link prediction problem for social networks," in *CIKM*, 2003, pp. 556–559.
[48] S. O'Mahony, "Hacking alone? the effects of online and offline participation on open source community leadership," September 2004.
[49] S. P. Borgatti, M. G. Everett, and L. C. Freeman, "Ucinet 6 for windows: Software for social network analysis," 2002, https://sites.google.com/site/ucinetsoftware/home.
[50] A. Usher, "The library for social network analysis," http://libsna.org/.