Dual Ecological Measures of Focus in Software Development

Daryl Posnett
UC Davis

Raissa D' Souza
UC Davis

Prem Devanbu
UC Davis

Vladimir Filkov
UC Davis

***Abstract*—Work practices vary among software developers. Some are highly *focused* on a few artifacts; others make wide-ranging contributions. Similarly, some artifacts are mostly authored, or *"owned"*, by one or few developers; others have very wide ownership. Focus & Ownership are related but different phenomena, both with a strong effect on software quality. Prior studies have mostly targeted ownership; the measures of ownership used have generally been based on either simple counts, information-theoretic views of ownership, or social-network views of contribution patterns. We argue for a more general conceptual view that *unifies* developer focus and artifact ownership. We analogize the developer-aritifact contribution network to a predator-prey food web, and draw upon ideas from ecology to produce a novel, and conceptually unified view of measuring focus and ownership. These measures relate to both cross-entropy and Kullback-Liebler divergence, and simultaneously provide two normalized measures of focus from both the developer and artifact perspectives. We argue that these measures are theoretically well-founded, and yield novel predictive, conceptual, and actionable value in software projects. We find that more focused, or specialized, developers introduce fewer defects than defocused developers. In contrast, files that receive narrowly focused activity are more likely to contain defects than those with a broader base of contributors.**

I. Introduction

Developers are the lifeblood of open source software. Their contribution is vital for open source software (*OSS*) to thrive. Rather than being assigned tasks by management, OSS developers are generally free to choose their style, focus and breadth of contributions. Some might be quite focused, working on one specific subsystem; others may range widely over the different subsystems. An expert in a particular hardware device, for example, might contribute very specialized knowledge to one or more open source projects. Such developers may focus on only a few files or a few packages to get a particular job done. That small subset of modules[1] may be their only contribution during their tenure with the project. In contrast, other developers may be more broadly focused, picking up a broad and varied range of open tasks across the project.

While OSS developers are free too choose their contribution sty-les, these choices are not inconsequential, especially to the central issue of *software quality*. A dominant theme emerging from previous work in this area is *module ownership*. If a module has many contributors or low ownership, this can adversely impact code quality. A variety of measures have been developed to measure ownership, *i.e.* the developers' contribution to a module [1], [2], [3].

There is, however, an entirely different perspective, *developer's attention focus*, which is relatively unexplored. Human attention and cognition are not unlimited [4]; mental resources, like other resources, are finite, and different tasks can compete for mental resources when simultaneously engaged, and task performance can suffer [5]. Programming work is arguably highly demanding and difficult. A developer engaged in many different tasks is likely to carry a much greater cognitive burden than one who is more focused. Interestingly, the developer and module perspectives are conceptually symmetric, dualistic views of **focus**. From a module's perspective, strong ownership indicates a strong focused contribution. Therefore, we refer to this as *module activity focus*, or $\mathcal{MAF}$. This is a measure of how focused the work activities on a module are. Symmetrically, we refer to the *developer's attention focus*, or $\mathcal{DAF}$. $\mathcal{DAF}$ is a measure of how focused the activities of a particular developer are. A surprising, but natural, analogy for $\mathcal{MAF}$ and $\mathcal{DAF}$ are predator-prey *food webs* from ecology. In a sense, modules are predators, which "feed upon" the cognitive resources of developers. The more different developers contribute to a module, the more different and diverse are the cognitive resources it "feeds" upon. Likewise a developer is a "prey" whose limited cognitive resources are spread over the modules that "prey upon" her.

The value of ecosystem diversity is of great interest to ecologists. Williams and Martinez assert "One of the most important and least settled questions in ecology concerns the roles of diversity and complexity in the functioning of ecosystems."[**?**]. At the community level, however, the diversity of predator-prey relationships are generally thought to be important to the survival of species, and ecosystem balance [**?**], [**?**]. This diversity has two symmetric perspectives, both from a prey's perspective, and a predator's perspective. Ecologists have developed sophisticated symmetric measures of predator-prey relationships, drawing upon ideas such as entropy and Kulback-Leibler divergence, that simultaneously captures both perspectives. We adapt these measures for software engineering projects, into our $\mathcal{MAF}$ and $\mathcal{DAF}$ metrics.

This novel analysis simultaneously considers focus both from the *artifact* perspective and the *author* perspective. Researchers can use our $\mathcal{MAF}$ and $\mathcal{DAF}$ metrics to more precisely evaluate how attributes of developer experience and focus contribute to outcomes of interest. Managers could also use these metrics to assess whether the degree of focus each

[1]We use modules to mean packages or files whenever the distinction isn't specifically relevant to the discussion.

developer exercises is in alignment with their expectations. For example, a manager may expect that the focus of a developer being groomed for a leadership role should decrease over time as he broadens his scope and focuses less energy on a particular portion of the code base. Similarly, a junior developer might be expected to observe a strong focus on particular modules. We employ methodology presented by El Emam to validate our measures [6]. In particular, we show that these measures succeed in distinguishing interesting cases that previously defined measures fail to capture. We make the following contributions:

- We adapt terminology and motivation from ecology, based on bipartite graphs;
- We incorporate and generalize previous results on developer and artifact diversity;
- We provide easy to compute measures of focus, $\mathcal{MAF}$ and $\mathcal{DAF}$, normalized to faciliate comparison within and across projects;
- We show that these measures more precisely capture outcomes of interest to software researchers and practitioners.

Research Outline Existing measures such as ownership and diversity only partially capture developer focus as we discuss in Section V. Consider, for example, device drivers. They are small but intricate, and will likely require "focused" work. If we measure the focus from solely the perspective of the module (the driver source code) we may be misled. Quite possibly, a single developer $\mathcal{D}$ contributed most of the coding activity in a driver module and traditional ownership measures will indicate that the driver has received focused activity. But, if $\mathcal{D}$ is a wide, prolific contributor, then the contribution she makes to the driver may not reflect focused attention. Indeed, $\mathcal{D}$ may have been distracted by many tasks, and the quality of the activity in the driver may be compromised.

Measuring focus solely from the developer's perspective is also insufficient. The attention of a particular developer may be highly focused on just a few files. However, if those activities make insignificant contributions, say just a few lines out of thousands, then we should describe her contributions as limited or minor, in comparison to those who contributed the bulk of the code. Given equal overall contributions, a developer whose attention is more focused on a small subset of the code base is viewed as exhibiting greater focused attention than the developer that contributes more uniformly across the entire system.

Our Goal: Simultaneously study the overall level and interplay of module activity focus and developer attention focus in OSS.

To that end we introduce the $\mathcal{MAF}$ and $\mathcal{DAF}$ measures in the Theory Section below. To understand them we mount a detailed study of module activity and developer attention foci in OSS projects. Starting with the combined perspective of both developer attention focus and module activity focus, we examine the descriptive statistics of these measures in projects; we also use multiple regression modeling to tease apart the effects of $\mathcal{MAF}$ and $\mathcal{DAF}$ on software quality and answer the following research questions.

Our measures enable us to characterize the attention focus of developers as broad or narrow. In particular, we ask if the leaders, or people with top involvement in open source projects are distinguishable by this measure?

Research Question 1: Do project developers exhibit broad or narrow attention focus? What about the top developers?

The effect of focus and collaboration on defects has been studied in the past. Some papers use social network models of developer collaborations; others use ownership measures of contributions by developers. Complex network measures such as degree centrality and betweenness are difficult to interpret and act upon. Ownership measures are better, but they ignore the effect of developer attention focus. Simultaneously modeling the diversity of both sides of the developer-module contributions, using $\mathcal{MAF}$ and $\mathcal{DAF}$, allows us to tease apart the effect of each one separately. As we shall see, both $\mathcal{MAF}$ and $\mathcal{DAF}$ can be acted upon based on the observed effects on software quality. First, from the developer perspective:

Research Question 2: Do developers with narrow attention focus create fewer defects?

The symmetry of our measures allows us to also ask the converse, from the *module* perspective:

Research Question 3: Are modules that receive narrowly focused activity less defective?

In summary, our goal here is to understand and generalize previous results on developer focus in software development contribution networks, realizing it is two-fold. We draw insights from similar conceptual structures that have been developed in ecology on specialization and expect that this will lead to more intuitively appealing, discerning, and actionable measures of contribution patterns. We begin by introducing some basic concepts: *contribution networks*, and *ecological networks*.

Contribution Networks Contribution networks model the total number of contributions made by each developer to each module over a specific period of time. Fig. 1(a) and Fig. 1(b) illustrate contribution networks as a bipartite network (having two kinds of nodes) consisting of modules and developers. Fig. 1(c) shows a network formed by the many-to-many relationships of numerous developers, each working on numerous modules. Social network analyses have been applied to these contribution networks [7], [8], using metrics largely derived from one-mode projections, which project the two-mode networks into either developers alone or modules alone. We use ecology-based measures of these networks, which

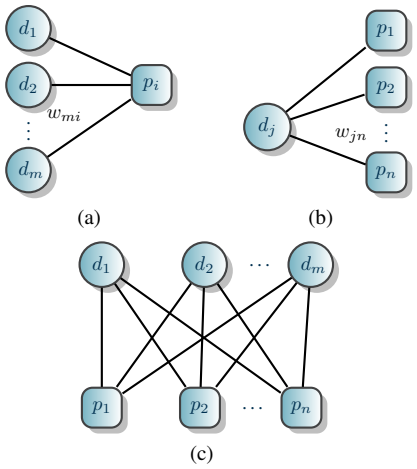

Fig. 1: Graphical representation of commits from multiple developers to a single package (a), from a single developer to multiple packages (b), and their bipartite representation (c).

preserve their essential bipartite nature.

Ecological Networks In the field of ecology, interaction networks relate predator to prey, pollinator to pollen, parasite to host, or simply organism to resource [9], [10], [11]. These networks usefully capture resource-consumption relationships, and the resulting effects on the relative abundance of different species in an ecosystem. They can, for example, help quickly identify species that are critical to ecosystems, or species whose survival is threatened.

In our context, the analogy gives some useful insights, but raises other questions. We can view artifacts as "consumers", and developers' resources as the "food source". Clearly, people's cognitive capacity is limited, and no one can work an unlimited amount of time. If a person's capacity is excessively spread out or diluted, two factors come into play. First, as diversity of an individual's ("food source") contribution targets ("consumers") increases, a given individual can only contribute a proportional amount of time to each of their foci (smaller portions of food); secondly, cognitive limitations (such as the difficulty of context-switching) come into play, and the quality of each contribution may go down [12]. From the other perspective, a particular module ("consumer") clearly benefits from contributions from a greater diversity of developers (more "food"). On the other hand, the contributions from each developer creates additional workload for the other contributors, as they must also understand these contributions, so additional contributors make everyone who "feeds" this module have to work harder (and thus provide less "food"). Thus, the straightforward analogy from food webs is complicated by cognitive limitations that introduce non-linear "interference" between contribution targets and contributors.

However, the above analysis-by-analogy with "food" and "consumption" highlights the two different perspectives on focus in contributor networks. The *developer* focus looks at how focused developers are in their contributions, and the *module* focus considers how focused the contributions to an artifact are. We frame our contributions by first presenting existing work in this area before we launch into the theory behind our new measures.

II. Related Work

Most previous work in this area has centered around the aggregation of ownership, largely considering the dominant author as a measure of artifact authorship. Our work most closely relates to recent work by Bird *et al.* [3], Rahman *et al.* [2], and by Mockus *et al.* [13]. These works study the relationship of quality to code ownership, from an *artifact* perspective. Bird *et al.* focus on *minor contributors*, *viz.*, those who contribute less than 5% of the content of an artifact, and finds that these play a strong role in defects. However, this perspective ignores the details of the contributions of these minor committers, and in fact is agnostic about the other activities of these contributors. What if a minor contributor d_1 contributed to a module f but didn't do anything else? She's a highly focused minor contributor; on the other hand, a minor contributor d_2 to f may in reality have worked on a great many other things. Bird *et al.* ignore this distinction; we in fact find that d_1 is less likely to produce defects than d_2.

Mockus *et al.* studies the risk of software changes by defining a measure that considers a developer's experience with the system as weighted by their experience with a particular modification request. Rahman *et al.* take a similar perspective, focusing on a developer's experience and ownership with a specific file as an indicator of quality of that file, regardless of the developer's other activities. Rahman *et al.* adapt the measures of Bird and Mockus; specialized experience measures the dominant contributors contribution to a particular artifact, and general experience is an adaptation of Mockus' weighted experience measure. The findings in Rahman *et al.* are generally consistent with Bird *et al.*, and have similar limitations.

Shannon's entropy was originally intended to quantify the information content in a signal. The idea of using entropy to measure properties of software evolution has a long history [14], [15] It has been used in numerous software engineering contexts; for space reasons we limit this discussion to some of the most recent efforts. Entropy has been applied to source code to measure the quality of interfaces [21], model readability [22], and measure the quality of modularization [23].

Recent work applying entropy to software systems has been driven by the wide availability of data from software repositories. Hassan and Holt proposed applying *normalized* entropy to a sliding time window of source code changes to capture the commit state of a project and used linear regression to predict defects in several open source projects [25]. That methodology has also since been improved and refined [26].

Canfora *et al.* used entropy to study the relationship between several factors, including refactoring, design patterns, and the number of contributers, and the entropy of changes as defined by Hassan and Holt [27]. The experiments yielded mixed results; refactoring changes showed a difference in means with respect to other changes in one of two projects, design patterns participation showed no relationship with entropy, and a general increase in the median entropy value was observed with increasing number of committers.

Taylor *et al.* introduced *author entropy* as discussed in the introduction [28]. Krien *et al.* extended the concept by defining author entropy across the distinct programming languages that a developer contributes code to within a project [29], [30]. Their analysis showed a clear negative relationship between language author entropy and lines of code contributed. Further studies have explored the use of entropy on both sides of the contribution network [31], [32].

Hindle *et al.* use topic analysis to study "what" a developer is focused on but their work does not capture the degree to which a developer is focused specific artifacts [**?**]. The idea of studying the contributions of individual developers to artifacts is not new. Pinzger *et al.* studied the effect of network metrics taken over the contribution network on software failures [7]. However, their results do not show significance of these measures for prediction of failure proneness even though they are significant in a linear regression model of defect counts. Consequently, it is difficult to derive any direct understanding of the relationship between the aforementioned measures and defect proneness [34]. Cataldo *et al.* build on earlier work in this area using network measures to more precisely define software and work dependencies.

Our work takes a unified approach to focus and ownership combining artifact- and developer-perspectives. This leads to important new findings. After we present our theory below we will discuss in more detail how our work builds upon these existing measures.

III. Theory

In this paper, we consider the relationships between developers and modules. For expository purposes, we use the term *module* in a the generic sense to represent a tangible unit of code relevant to the research question. *Module* could refer to a file, a package, a component, or even simply a function.

An individual developer is denoted by d_j, for $j \in 1 \dots m$ and an individual module by m_i, for $i \in 1 \dots n$. The total contribution count to module i by developer j, is denoted by w_{ij}. Summing over all developers and all modules yields the total number of contributions to the system as $A = \sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij}$. We can also calculate the total number of contributions that developer j makes to the system by $D_j = \sum_{i=1}^{n} w_{ij}$. Similarly, the total number of contributions made to module i by all developers is denoted by $M_i = \sum_{j=1}^{m} w_{ij}$. Our specific measure is conceptually based on Shannon's entropy, which measures the disorder or surprise (*viz.*, information) in a system. The aforementioned "author entropy" [28] due to Taylor is precisely Shannon's entropy over a probability distribution based on the weight of contributions from each developer.

$$H_i = -\sum_{j=1}^{m} p_{ij} \log_2 p_{ij}$$

where $p_{ij} = {}^{w_{ij}}/_{M_i}$. The contribution w_{ij} can be measured in lines of code, commits, or any other measure of contribution relevant to the studied context. In this work we measure proportion of contribution as the number of commits contributed by each author.

Diversity The definition of author entropy given by Taylor *et al.* is precisely the concept of diversity from ecology. Theoretical ecologists were among the first to employ Shannon's entropy as a measure of diversity in a species [35]. A straightforward explanation of what ecologists mean by diversity can be found in a recent discussion by Camargo [36]. We summarize here for the purposes of clearly translating the intuition to a software engineering setting. Camargo presents the definition as follows:

$$H(\text{Species Diversity}) = -\sum_{i=1}^{S} p_i \log_2 p_i$$

Here S represents the total number of distinct species and p_i is the proportion of individuals that are of species i. If we compare this to the definition due to Taylor *et al.* we can see that where ecologists refer to *species* and *individuals*, Taylor refers to *developers* and *ownership*. If all species are equal in number, then diversity is high, if one particular species dominates in number then the diversity measure will be low. With respect to *author entropy*, however, diversity simply measures the uniformity of the commits relative to each author. If one author makes most of the commits, then low diversity is taken to mean that one author dominates the commit activity. Similarly, we can also consider the diversity of authorship commit activity with respect to a module. A module that has only one author is not diverse at all and a package with many authors sharing the load has a high degree of diversity. Diversity here is simply the diversity of commit behavior for either authors over a module, or modules over a particular author.

Using the notation presented above (and also in Fig 1) one can formulate *two distinct definitions* of diversity, from two different perspectives: one with respect to the contributions of author d_j to all modules H_{d_j}, and the other with respect to the contributions to a module m_i from all authors H_{m_i}, as follows:

$$H_{d_j} = -\sum_{i=1}^{n} \left(\frac{w_{ij}}{D_j} \ln \frac{w_{ij}}{D_j} \right) \quad , \quad H_{m_i} = -\sum_{j=1}^{m} \left(\frac{w_{ij}}{M_i} \ln \frac{w_{ij}}{M_i} \right)$$

Specialization & Focus Specialization in a general sense is the opposite of diversity; the more specialized a developer's behavior, the less diverse is his contribution to a project. This property, proposed by Bluthgen *et al.* [37] in an ecological setting, can be measured naturally in the bipartite graph formulation described above. To distinguish our use from the terminology in ecology, and to better reflect the actual cognitive phenomena of concern in software development, we prefer the term *focus*. So, then why not just use the above mentioned diversity measures for this purpose, applying them to each dimension in the contribution network to measure focus? As Bluthgen *et al.* point out, this approach is undesirable in an ecological setting; their arguments also apply in a software development setting.

An appropriate measure of focus in software development should not only consider the diversity of artifacts that a developer interacts with, but also, the overall amount of activity that

those artifacts are subject to. A developer who only commits a few times to a popular package with many commits, is *less* specialized than a developer who makes similar commits to an unpopular package. A good measure of focus, then, should increase when a developer makes most of the contributions to a package when compared with others who contribute to that package. To accomplish this, we want to measure the difference between the distribution of commits made by a developer to all modules and the distribution of commits to the system represented by those modules.

Kullback-Liebler Divergence Kullback Liebler divergence, or relative entropy, measures the difference between two probability distributions. For probability distributions P and Q the Kullback Liebler Divergence (*KL*) is defined as:

$$D_{KL}(P||Q) = \sum_i \ln \frac{P_i}{Q_i}$$

KL is a measure of the expected number of extra bits that are required to code samples from P when using a code based on Q. Bluthgen *et al.* define a species level diversity measure, d, using the Kullback-Liebler Divergence. We exploit this measure in our context to relate our two probability distributions of interest.

Our Measures: $\mathcal{DAF}$ and $\mathcal{MAF}$ We introduce two measures: *Developer Attention Focus* , or $\mathcal{DAF}$, measures the divergence from the developer perspective, *viz.*, the degree of focus a developer exercises with respect to the artifact side of the network. From the artifact side, *Module Activity Focus* , or $\mathcal{MAF}$, measures the degree to which a module receives focused attention. We define the proportion of commits made by developer j to module i as $q'_{ij} = {}^{w_{ij}}/_{D_j}$ and the proportion of commits made to each module i as $r'_{ij} = {}^{w_{ij}}/_{M_i}$. The total proportion of commits made to each package is $r_i = {}^{M_i}/_{A}$ and the total proportion of commits made by each developer is $q_i = {}^{D_j}/_{A}$.

We adapt the Bluthgen *et al.* notation to be compatible with ours and substitute δ_j and δ_i in place of d to avoid confusion and to convey clearly which side of the network each metric is associated with. This (un-normalized) measure compares the distribution of the interactions with each network partner, *viz.* developers and modules, to the overall partner contribution.

$$\delta_j = \sum_{i=1}^{n} \left(q'_{ij} \ln \frac{q'_{ij}}{r_i} \right), \qquad \delta_i = \sum_{j=1}^{m} \left(r'_{ij} \ln \frac{r'_{ij}}{q_j} \right)$$

$\mathcal{DAF}$ Intuition The intuition behind these measures is straightforward, which is seen clearly from a small transformation; using δ_j (un-normalized $\mathcal{DAF}$) as an example we obtain the following.

$$\begin{aligned}
\delta_j &= \sum_{i=1}^{n} \left(q'_{ij} \ln q'_{ij} - q'_{ij} \ln r_i \right) = \sum_{i=1}^{n} q'_{ij} \ln q'_{ij} - \sum_{i=1}^{n} q'_{ij} \ln r_i \\
&= \left(- \sum_{i=1}^{n} q'_{ij} \ln r_i \right) - \left(- \sum_{i=1}^{n} q'_{ij} \ln q'_{ij} \right) \\
&= \left(- \sum_{i=1}^{n} \frac{w_{ij}}{D_j} \ln \frac{M_i}{A} \right) - \left(- \sum_{i=1}^{n} \frac{w_{ij}}{D_j} \ln \frac{w_{ij}}{D_j} \right)
\end{aligned}$$

This measure is computed for each developer, *viz.* we fix j, and it is computed over all modules. There are two terms in the δ_j equation, which merit separate explanation.

The left term is the *cross entropy* of developer j's contributions with the module level contributions from all developers. The intuition is that if the proportion of a developer's contributions to each module are similar in distribution to the proportion of commits to each module overall, then we are unsurprised; the developer's contributions mimic the project as a whole. He works less on modules that are just a small part of the overall system and works a great deal on the more substantial modules. On the other hand, if the distributions are dramatically different, then the developer's contribution is unexpected, increasing cross entropy, and hence, the focus metric. If, *e.g.*, he works solely on modules that comprise just 25% of the system, then his contributions are out of proportion and should be seen as focused.

The right term corrects for a complication. Suppose that the above mentioned 25% of the system is spread out over many tiny modules that each represent a small fraction of the system. Then, even though our developer's contributions are disproportionate (limited to just 25% of the system), they are actually quite scattered, and we would not consider him so focused; instead he is rather distracted. In this case we want to penalize his focus score to reflect this distraction. This is accomplished by the right term, which is simply developer diversity as described in the previous section. The more spread out a developer is, the higher his diversity score. In summary, $\mathcal{DAF}$ is high when a developer both monopolizes packages and is not distracted by too many other packages.

$\mathcal{MAF}$ Intuition The derivation of $\mathcal{MAF}$ mirrors $\mathcal{DAF}$, so we present only the final line to aid understanding.

$$\delta_i = \left(- \sum_{j=1}^{n} \frac{w_{ij}}{M_i} \ln \frac{D_j}{A} \right) - \left(- \sum_{j=1}^{n} \frac{w_{ij}}{M_i} \ln \frac{w_{ij}}{M_i} \right)$$

This measure is taken from the complementary side of the network and is computed for each module, *viz.* we fix i, and compute the metric over all developers. While the computation is virtually identical, it captures a different aspect of the contribution network.

The left term is the *cross entropy* of the contributions to the module, with the developer contributions to the system. As before, if the distributions are similar, then the module receives minimal contributions from developers who are not major contributors to the overall system, and sufficient attention from those developers most responsible for the system. In this case, the module's focus is low, and we're not particularly surprised. For example, if a `README` file has received a fraction of a percent of the commits from a particular developer, we're not surprised so long as the file isn't dominating that developer's attention. If, in fact, it is dominating, then this excess attention is *focused attention* to the module. Similar to $\mathcal{DAF}$, the right term penalizes the focus for the diversity of contributions. Modules equally contributed to by multiple developers do get less focused attention than modules with high ownership. In summary, $\mathcal{MAF}$ is high when a package

monopolizes a developer's attention and receives little attention from other developers.

$\mathcal{MAF}$ and $\mathcal{DAF}$ are both normalized by the theoretical maximum and minimum possible values of the measures. For the max, $\delta_{j_{max}} = ln\ {}^{A}/_{D_j}$ and $\delta_{imax} = ln\ {}^{A}/_{P_i}$. The theoretical minimum value of 0 is typically not attainable in the case where the proportional counts are based on integer values, as is the case here, so a heuristic is used to find a suitable minimum (See [37] for more detail)[2]. Using these minimum and maximum values the δ_j is standardized to a 0 to 1 range with the following normalization.

$$\mathrm{DAF}_j = \frac{\delta_j - \delta_{j_{min}}}{\delta_{j_{max}} - \delta_{j_{min}}}, \qquad \mathrm{MAF}_i = \frac{\delta_i - \delta_{i_{min}}}{\delta_{i_{max}} - \delta_{i_{min}}}$$

Since these measures specifically take into account the contributions that each developer makes and that each module receives, their values are independent of this variation within a network and can be used to compare the relative focus levels of individual developers and modules.

Each metric can be interpreted as a deviation of contribution frequencies from a null model which assumes that all developer/module pairs are contributed from/to in proportion to the overall contribution to the system. In the simplest case of a fully balanced network where all $q'_{ij} = r'_{ij}$ the theoretical minimum value of 0 will be achieved. When $q'_{ij} = r'_{ij}$ then the cross entropy is indistinguishable from the entropy and, whatever value they attain, their difference is 0 and focus is minimized.

In Section V we discuss how our metrics related to existing work and present a small case study on a real system to help illuminate how it is able to distinguish interesting cases. In the next section we describe the data and methods used for the case study and for the statistical analysis presented in Section VI.

IV. Data and Methodology

Data We extracted a selection of metrics often used for defect prediction for seven projects maintained by the Apache Software Foundation, listed in Table I. For each project we used data from the source code repository and the Jira issue tracking system to extract basic process metrics such as churn and the number of commits and devs associated with each file and package. The specific metrics used are described breifly with each model description.

Project	Releases	# Files	# Packages
Avro	1.3.2 - 1.4.1	158-238	12-17
Cassandra	0.6.0 - 0.6.8	314-332	31-33
CXF	2.11-2.3.1	3086-4097	491-598
Ivy	2.0.0 - 2.2.0	481-498	65-67
Lucene	1.9.1 - 3.0.3	1010-957	102-85
Shindig	2.0.0, 2.0.1	811-812	75-75
Wicket	1.2.7 - 1.3.7	1776-1947	240-249

TABLE I: Apache Software Foundation projects and their description

[2]Also the R bipartite package, http://cran.r-project.org/web/packages/bipartite/index.html

Jira is an issue and bug tracking system that manages a database of issue reports submitted by developers and users. Issues can be of various types including new features, improvements, or defects. Jira enforces a basic development process by mapping issue reports to version control commit messages. It accomplishes this by cross linking Jira issue IDs extracted from version control system commit log messages with the associated report in the Jira database. We extracted the Jira issues from the XML report available on the Apache Software Foundation's project website for each of the projects.

Version Control Version control systems, *e.g.* Git, SVN, and CVS, facilitate collaboration among developers by maintaining a history of changes and an associated log entry for each change. To obtain the number of commits and developers associated with each file we parse the log data retrieved from the version control system. We use the Jira issue IDs and the Git version control log to link issues associated with each commit to the files involved in the commit to associate defect counts with each file in the release.

Bug-Introducing Change For defect issues in the Jira database we'd like to try to locate the files that induced the defect fixing modification. The lines of code associated with the changes that triggered such a modification are referred to as "fix inducing code" as coined by Sliwerski *et al.* [39].

To identify the fix inducing code we use the *SZZ* algorithm developed by Sliwerski-Zimmerman-Zeller [39]. We extract commits associated with each defect fix as described above. If a fix is associated with revision n then we apply *git diff* to revision $n-1$ and revision n to identify the specific lines that were changed in the fix. We then use *git blame* on only the changed lines to identify the revision responsible for the fix inducing code. If the revision was changed after the defect introduction then we do not associate post defect changes with the defect, otherwise we associate the unique defect ID with the file in the fix inducing revision. This allows us to identify the unique, modulo SZZ accuracy, defects that can be blamed on each file in the system.

Contribution Networks We described contribution networks in Section I. To gauge the global focused attention of each developer over the life of the project we built a network using all entries in the commit graph over the full period of each project that we studied. This approach results in static focused attention values for each developer and for each module.

V. Analysis

With respect to the methodolgoy described by El Emam we want to show that our new metrics $\mathcal{MAF}$ and $\mathcal{DAF}$ capture interesting properties not captured by other metrics [6]. We discuss this in an analytical setting first and conclude this section with a case study. In Section VI we strengthen this analysis with some statistical results.

Ownership The simplest measure of focus is module ownership which is usually measured as the proportion of contributions to a module. The developer with the highest ownership is identified as the module's owner. Not surprisingly, ownership shows no meaningful correlation with $\mathcal{MAF}$ but strong correlation with both betweenness measures (see Sec. V); trivially,

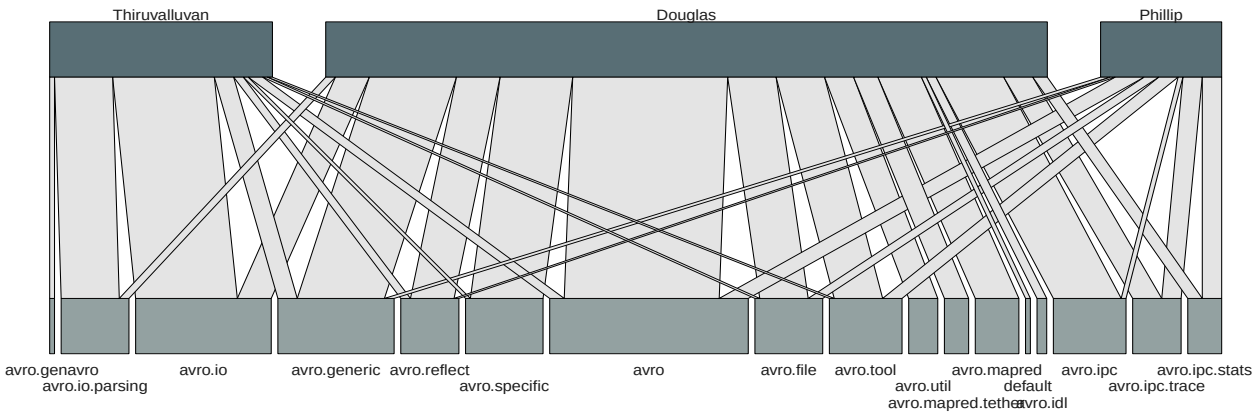

Fig. 2: The full bipartite network of Apache Avro with developers at the top and files on the bottom. Block and edge sizes represent the number of commits either from, or to, developers and packages respectively.

ownership cannot capture any significant nature of developer or module interaction as the measure itself considers only a single developer and module in isolation.

Betweenness In prior research, betweenness in contribution networks has been used as a proxy for developer focus. It is a simple measure of module connectivity, measuring the fraction of geodesic paths between developers that pass through that module [38]. So if a module was touched by fewer developers who themselves touched few other modules, then betweenness will be low indicating high focus. We also consider the "developer network edge betweenness" (*DNBetweenness*), a collapsed view where developers are the nodes of the graph and the modules connecting nodes form the edges. Betweenness measures are often highly correlated with the number of developers. As more developers touch a file, the likelihood of betweenness goes up; so, however, does the number of commits, the size of the code, the number of bug fixes, and consequently, the number of defects (see Table III). Thus, betweenness, in practice, rarely adds any predictive or actionable value beyond the raw developer count.

Diversity is a more sensitive measure that represents the commit entropy of each module. Like betweenness, high diversity values implies a lack of contribution focus. Low values of diversity suggest low surprise at who is going to commit next. Diversity doesn't distinguish cases where this surprise might be important, *e.g.* a device driver, with cases where it is largely irrelevant, *e.g.* a `README` file.

Experience Some measures of experience consider the contributions that a developer has made to the system along with his contributions to an entity; specifically, Mockus' experience measure, although applied in a different setting, is the closest to ours [13]. In fact, it can be shown that this measure is equivalent to a normalization term minus the cross entropy $H(r'_{ij}, q_j)$. It does not, however, fully capture the nature of developer "focus" as it does not capture diversity, does not have a standard range, and most importantly, does not define a developer focus metric such as $\mathcal{DAF}$.

Additionally, it is worthwhile to note that the dual nature of our measures permits an elegant computation, *viz.*, that the same code can be used to calulate either measure on the weighted adjacency matrix that represents the contribution network simply by transposing the matrix to obtain the dual.

	Ownership	Diversity	Focus $\mathcal{MAF}$	Btw	DN Max Btw.	Bugs
avro	0.78	0.67	0.04	0.12	1	1
avro.file	0.71	0.76	0.03	0.12	1	2
avro.specific	0.94	0.23	0.09	0.12	0	1
avro.tool	0.67	0.80	0.05	0.12	1	0
avro.ipc.stats	0.57	0.68	0.21	0.00	0	0
avro.io	0.75	0.56	0.34	0.12	0	0
avro.io.parsing	0.86	0.41	0.36	0.12	0	1
avro.genavro	1.00	0.00	0.23	0.00	0	0
avro.generic	0.75	0.72	0.01	0.12	1	4
avro.reflect	0.75	0.72	0.00	0.12	1	0
avro.util	1.00	0.00	0.10	0.00	0	0
avro.ipc	0.93	0.24	0.10	0.00	0	2
avro.mapred	1.00	0.00	0.12	0.00	0	0
avro.mapred.tether	1.00	0.00	0.10	0.00	0	0
avro.idl	1.00	0.00	0.02	0.00	0	0
avro.ipc.trace	0.60	0.67	0.14	0.00	0	0
default	1.00	0.00	0.00	0.00	0	0

TABLE II: Detailed attributes of packages in Apache Avro.

Avro Case Study Apache Avro is a remote procedure call and serialization framework. It's primary purpose is to provide serialization for persistent data and a format for distributed data exchange. It is a moderately-sized project with over 200 files. We gathered 110 distinct commits of the project between Jan 2010 and Sep 2010. During that time there were only three committers to the project. Because of this small size we use it here to illustrate some properties of various focus metrics.

The full bipartite package level contribution network can be seen in Fig. 2. Block and edge sizes in the graph represent the number of commits either from, or to, developers and modules respectively. We first consider $\mathcal{MAF}$ in concert with related measures and then $\mathcal{DAF}$.

$\underline{\mathcal{MAF}}$ Of interest here are the previously defined entropy measures, the simple proportional measure of ownership, and the concepts of node and edge betweenness.

Ownership is the simplest to discuss; consider package `avro` in Fig. 2 which is owned by Douglas with ownership of 0.78. With a $\mathcal{MAF}$ score of 0.04 we might conclude that ownership and $\mathcal{MAF}$ are inversely related. However `avro.specific` has both higher ownership, and higher $\mathcal{MAF}$. Table III shows that $\mathcal{MAF}$ and ownership are not strongly correlated. Betweenness considers more of the network, but, since betweenness measures, as in Meneely *et al.* [8], do not take into account the relative contribution of each developer, they are often highly correlated with the number of developers and other "size" metrics. Further, because in this case the developers are tightly connected by the modules, neither form of betweenness discriminates interesing cases.

The packages`avro.file` and `avro.specific` have identical betweenness values of 0.125 indicating that they are unfocused artifacts. Their diversity values, however, lie at either end of the spectrum. A total of three developers have contributed to `avro.file` which has a diversity score of 0.759. The width of the ribbon connecting Phillip to this package indicates that he has made a modest but significant proportion of the commits to the package. In addition Thiruvalluvan has also made at least one commit, although his contribution is smaller. Douglas has made most of the commits, but even so, the total number of contributions is still relatively small. The package `avro.specific`, on the other hand, has only two contributors and Douglas clearly owns the lion's share resulting in a much lower diversity value of 0.234, indicating the relative lack of surprise at who will make the next commit.

In the previous example, diversity wass able to capture variance in contribution that betweenness cannot. Unfortunately, diversity is insufficient to capture many examples of focus or specialization. `avro.ipc.trace` and `avro` have almost identical diversity values of 0.673 and 0.683 respectively. While their betweenness values are different, again, so are their number of developers. Package `avro.ipc.stats` also has a similar diversity score of 0.683 and like `avr.ipc.trace` has only two developers, thus, its betweenness value is 0. In this example, $\mathcal{MAF}$ is able to discriminate between many interesting cases that other metrics fail to capure.

	$\mathcal{DAF}$	Diversity	#Commits	Bug Fixes	Bugs
Douglass	0.23	2.48	71	26	7
Thiruvalluvan	0.51	1.57	25	11	2
Philip	0.30	1.94	14	2	2

$\underline{\mathcal{DAF}}$ From the contribution network we can see at a glance that Thiruvalluvan probably has the greatest focus. Most of his commits are focused in two packages whereas both Douglas and Philip do not appear to have any packages that dominate their contribution patterns. This distinction is important; clearly Douglass dominates the `avro` package and one can view the package as receiving focused contributions, however, it does not dominate Douglas' contributions to the system. Even though it is the package that Douglas has contributed to the most, it has received, at most, approximately twice the contributions of the next smaller package and there are at least three or four such packages. Compare with Thiruvalluvan who's largest contribution accounts for about half of his total contributions to the system. In this instance, developer diversity *(author entropy)* captures this difference nicely ranking Thiruvalluvan as the most focused developer and Douglas as the least focused.

	MAF_i	Betweenness	DNMaxBetweenness
MAF_i	1.00	-0.19	0.05
Ownership	-0.07	-0.47	-0.49
Diversity	-0.18	0.66	0.85
Betweenness	-0.19	1.00	0.70
DNBetweenness	0.05	0.70	1.00
Bugs	-0.19	0.49	0.53

TABLE III: Spearman rank correlation of the measures discussed in the text

Consider again packages `avro.file` and `avro.specific`, which have identical betweenness values and drastically different diversity values, the $\mathcal{MAF}$ values are both fairly low at 0.034 and 0.087 respectively. The skewed contribution from Douglas yields the low diversity score indicating high focused attention, but the contribution from Douglas does not represent a disproportionate focus from his perspective. It cannot be argued that Douglas is focused on this package, his efforts are spread out fairly evenly over a number of packages. So $\mathcal{MAF}$ and $\mathcal{DAF}$ measures capture focus from both the perspective of the developer and the module. A focused package is one that receives a lot of attention from few developers who devote most of their attention to that package.

Interestingly, the two packages with the highest focused activity scores are `avro.io.parsing` and `avro.io`. Both of these packages are dominated by Thiruvalluvan. By simply looking at the web we might conclude that Thiruvalluvan is something of an I/O specialist, that Douglas is the project leader, and that both he and Phillip exhibit broader, less focused attention. Both the $\mathcal{MAF}$ and $\mathcal{DAF}$ scores, which are mostly low with the exception of Thiruvalluvan and his focus on I/O, support this hypothesis.

VI. Results and Discussion

We show in Fig. 3, left, the distribution of $\mathcal{DAF}$ over all projects of with respect to files. While $\mathcal{DAF}$ exhibits distinct means and variances for different projects, there is no apparent correlation with project size (viz. TableI). The right hand side of Fig. 3 shows the distribution of $\mathcal{MAF}$. Mean developer specialization is around 0.5 and the variance is project dependent. The mean $\mathcal{MAF}$ scores are much lower, approximatelyl 0.10 with respect to files. Unlike $\mathcal{DAF}$, however, the distributions are similar across projects. This to

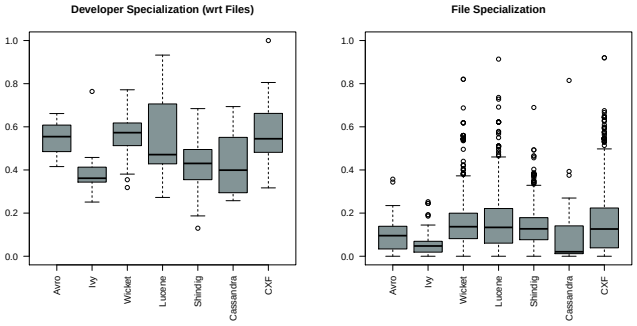

Fig. 3: $\mathcal{DAF}$ (on left) exihibits greater mean and variance across all projects than $\mathcal{MAF}$ (on right)

be expected as developers touch more files, than files are touched by developers as a result of the assymetry in the collaboration network.

For the remaining research questions we used *negative binomial regression, NBR* to model count data against some parameters of interest. NBR is a generalized linear model used to model non-negative integer responses. It is appropriate technique here as it can handle *over dispersion*, *e.g.*, where the variance is greater than the mean, in the response [40]. For this study our focus is on understanding the mean within project behavior so we view each project as a random effect in a pooled model incorporating it as a grouping factor. The project factor is used to capture the between project variance in the response.

RQ1: Overall and top-contributor attention focus

To determine the level of contribution by focused devs we regressed the number of commits against the number of files touched and a developers $\mathcal{DAF}$ score. A manual examination of the data revealed that the top 5% of contributors touch significantly more files than the remaining developers necessating the inclusion of files in the regression model. We can see from the model that the $\mathcal{DAF}$ coefficient is only significant at a 10% level after controlling for the number of files touched.

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	3.3642	0.3775	8.91	$< 2e-16$
$\mathcal{MAF}$	-1.0528	0.6339	-1.66	0.0967
files	0.0072	0.0006	11.42	$< 2e-16$

So while focused developers contribute to fewer files, even will controlling for the confound, they do not contribute as much code as more broadly focused developers.

	Ivy	Avro	Wicket	Shindig	Lucene	Cassandra	CXF
$\mathcal{DAF}$	0.17	0.23	0.20	0.08	0.19	0.09	0.14
$\overline{\mathcal{DAF}}$	0.41	0.54	0.57	0.41	0.56	0.43	0.57

TABLE IV: Dominant project contributor levels of specialization.

As we saw with the Avro project, it's often the case in open source that there is a dominant project contributor. We might expect that project leaders exhibit lower attention focus if they contribute a significant proportion of the project's code. As a developer's contribution increases, it becomes increasingly difficult to contribute a greater proportion of code than expected to the many files currently touched. The dominant contributor for each project is listed in Table IV. In each case the attention focus scores of the dominant contributors are below the mean.

> **Result 1:** *Project leaders and top contributors tend to exhibit lower attention focus than others. The affect of attention focus on contribution by the top contributors is significant at the 10% level after controlling for the number of files changed.*

RQ 2: Do narrowly focused developers create fewer defects?

To answer this question we regressed the file contribution pattern of each developer against the number of defects introduced by that developer. We are interestd in the degree to which focused attention could explain their contribution of defects to the software over its lifetime.

For this experiment we evaluated the entire contribution network for each project obtaining a mean attention focus score for each developer over the life of the project. We modeled the total number of induced defects attributed to each developer against the developers file attention focus scores. We limited the number of size-based control variables (which are strongly inter-correlated) to avoid a high variance inflation factor *VIF* [40]. We included the number of files to control for the spread of developer focus simply based on the number of artifacts changed. Similarly, we include the number of commits to control for the positive relationship between the number of changes made and the number of defects.

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	3.7249	0.2407	15.48	$< 2e-16$
$\mathcal{DAF}$	-2.4462	0.1792	-13.65	$< 2e-16$
files	0.0024	0.0002	15.14	$< 2e-16$
commits	-0.0012	0.0001	-9.25	$< 2e-16$

As can be seen from the above model details, after controlling for the number of files as well as the number of changes, $\mathcal{DAF}$ has a negative effect on the number of defects induced by a developer, i.e. the more narrowly focused the developer, the fewer defects that will be introduced. Our results at the package level were quite similar which suggests that this relationship is robust to ecological inference risk [**?**].

> **Result 2:** *Narrowly focused developers introduce fewer defects at both the file and package level*

RQ 3: Do files that exhibit narrowly focused activity have fewer defects?

It follows that if developers with narrow focus introduce fewer defects then we might find that files that are the center of narrow activity focus, *viz.*, that a low $\mathcal{MAF}$, would lead to fewer defects.

Project	Regressor	Estimate	Std. Error	z value	Pr(>|z|)
	(Intercept)	-4.1503	0.2499	-16.61	¡ 2e-16
	$\mathcal{MAF}$	0.9430	0.2647	3.56	0.00037
	ownership	-0.3398	0.1901	-1.79	0.07396
	log(commits)	0.8733	0.0618	14.13	¡ 2e-16
	log(devs)	0.1697	0.0916	1.85	0.06385
	log(loc)	0.2261	0.0290	7.81	5.7e-15

TABLE V: Negative binomial model detail for for the affect of δ_i' specialization and ownership against defects in files. The model includes controls for the number of developers, the number of commits, and the lines of code. $\mathcal{MAF}$ is significant after controlling for the number of changes and is postively associated with the number of defects in files

Previous results support focused attention to files by linking file based focus metrics to defects. We also considered this question regressing $\mathcal{MAF}$ and other file properties against the number of defects induced in each file. Here we control for file, size, the number of commits, and the number of developers.

We included ownership in the model to test whether this simple measure was able to capture a similar relationship with defects.

The details of the model is shown in Table V. Surprisingly, the direction of the coefficient is positive. In other words, increasing focused activity has a *negative* impact on software quality while holding other factors constant. When considered in concert with our RQ4 this suggests that while it is important for a developer to focus his efforts to avoid excessive unfocused contribution, it is also important for files to receive some general attention. We also not that, increased ownership is negatively correlated with defects which is in agreement with previous results, however, is not significant at the 5% level.

There could be several factors driving this result. Files with high $\mathcal{MAF}$ may be more complex and are consequently naturally more defect prone. It may also simply be the case that if attention is too focused in a file then this indicates that greater diversity is necessary in order that defects are found.

Result 3: *Files narrow focused activity at the file level results in a greater number of defects.*

VII. Conclusion

The specialization measures we introduced have roots in ecology but are very well suited for analysis of focus and focus change in a symmetrical setting between developers and the artifact and artifact behavior both globaly and localy. In addition, specialization used in modeling allows for easy interpretation of results. E.g., the effect of focus on defect introduction is particularly clear: when developer focus is higher there are fewer defects introduced. With files, the effect of their focus on defects is clearly not strong. This is a welcome side-effect of the artifact/developers symmetry of this measure: it allows for deconvolving the joint effects into separate contributions of the artifact and the developers, so they can be considered and reasoned about individually.

Threats To Validity We recognize a few threats. Complex files do get more defects. It would be revealing to study the relationship between code complexity and focus doled out or received. However, since file complexity is typically correlated with code length, and our models do take length into account, the effects of file complexity on our results is likely small. Being that our measures are based on information theory they inherit the associated benefits and challenges. In particular, information theoretic measures are well known to necessitate substantial amounts of data to yield appropriate results.

Finally, the focus measure can be useful in practice as an assesment tool, as defocus with time is correlated with defects introduced. Sound coding practices could incorporate this measure.

References

[1] E. Weyuker, T. Ostrand, and R. Bell, "Do too many cooks spoil the broth? using the number of developers to enhan ce defect prediction models," *Empirical Software Engineering*, vol. 13, no. 5, pp. 539–559, October 2008. [Online]. Available: http://dx.doi.org/10.1007/s10664-008-9082-8

[2] F. Rahman and P. Devanbu, "Ownership, experience and defects: a fine-grained study of authorship," in *Proceeding of the 33rd international conference on Software engineering*. ACM, 2011, pp. 491–500.

[3] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Dont touch my code! examining the effects of ownership on software quality," in *Proceedings of the the eighth joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering. ACM*, 2011.

[4] R. Baumeister, E. Bratslavsky, M. Muraven, and D. Tice, "Ego depletion: Is the active self a limited resource?" *Journal of personality and social psychology*, vol. 74, no. 5, p. 1252, 1998.

[5] S. Aral, E. Brynjolfsson, and M. Van Alstyne, "Information, technology and information worker productivity: Task level evidence," 2007.

[6] K. El-Emam *et al.*, "A methodology for validating software product metrics," 2000.

[7] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer-module networks predict failures?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, 2008, pp. 2–12.

[8] A. Meneely and L. Williams, "Secure open source collaboration: an empirical study of linus' law," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 453–462.

[9] R. Arditi and L. R. Ginzburg, "Coupling in predator-prey dynamics: Ratio-dependence," *Journal of Theoretical Biology*, vol. 139, pp. 311–326, 1989.

[10] R. J. Williams and N. D. Martinez, "Simple rules yield complex food webs," *Nature*, vol. 404, pp. 180–183, 2000.

[11] M. Pascual and J. A. Dunne, Eds., *Ecological networks: linking structure to dynamics in food webs*. New York: Oxford Univ. Press, 2006.

[12] H. Pashler, "Task switching and multitask performance," *Attention And Performance*, pp. 277–307, 2000.

[13] A. Mockus and D. Weiss, "Predicting risk of software changes," *Bell Labs Technical Journal*, vol. 5, no. 2, pp. 169–180, 2000.

[14] M. Emden, "On the hierarchical decomposition of complexity," 1969.

[15] R. Chanon, "On a measure of program structure," in *Programming Symposium*. Springer, 1974, pp. 9–16.

[16] S. Mohanty, "Models and measurements for quality assessment of software," *ACM Computing Surveys (CSUR)*, vol. 11, no. 3, pp. 251–275, 1979.

[17] S. Henry and D. Kafura, "Software structure metrics based on information flow," *Software Engineering, IEEE Transactions on*, no. 5, pp. 510–518, 1981.

[18] N. Chapin, "An entropy metric for software maintainability," in *System Sciences, 1989. Vol. II: Software Track, Proceedings of the Twenty-Second Annual Hawaii International Conference on*, vol. 2. IEEE, 1989, pp. 522–523.

[19] W. Harrison, "An entropy-based measure of software complexity," *Software Engineering, IEEE Transactions on*, vol. 18, no. 11, pp. 1025–1029, 1992.

[20] G. Visaggio, "Structural information as a quality metric in software systems organization," in *Software Maintenance, 1997. Proceedings., International Conference on*. IEEE, 1997, pp. 92–99.

[21] O. Panchenko, S. Mueller, and A. Zeier, "Measuring the quality of interfaces using source code entropy," in *Industrial Engineering and Engineering Management, 2009. IE&EM'09. 16th International Conference on*. IEEE, 2009, pp. 1108–1111.

[22] D. Posnett, A. Hindle, and P. Devanbu, "A simpler model of software readability," in *Proceeding of the 8th working conference on Mining software repositories, MSR*, vol. 11, 2011, pp. 73–82.

[23] S. Sarkar, A. Kak, and G. Rama, "Metrics for measuring the quality of modularization of large-scale object-oriented software," *Software Engineering, IEEE Transactions on*, vol. 34, no. 5, pp. 700–720, 2008.

[24] A. Hassan and R. Holt, "The chaos of software development," in *Software Evolution, 2003. Proceedings. Sixth International Workshop on Principles of*. IEEE, 2003, pp. 84–94.

[25] ——, "Studying the chaos of code development," in *Proceedings of the 10th Working Conference on Reverse Engineering*. IEEE Computer Society, 2003, p. 123.

[26] A. Hassan, "Predicting faults using the complexity of code changes," in *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 2009, pp. 78–88.

[27] G. Canfora, L. Cerulo, M. Di Penta, and F. Pacilio, "An exploratory study of factors influencing change entropy," in *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on*. IEEE, 2010, pp. 134–143.

[28] Q. Taylor, J. Stevenson, D. Delorey, and C. Knutson, "Author entropy: A metric for characterization of software authorship patterns," in *Third International Workshop on Public Data about Software Development (WoPDaSD08)*, 2008, p. 6.

[29] J. Krein, A. MacLean, D. Delorey, C. Knutson, and D. Eggett, "Language entropy: A metric for characterization of author programming language distribution," in *4th Workshop on Public Data about Software Development*, 2009.

[30] J. Krein, A. MacLean, C. Knutson, D. Delorey, and D. Eggett, "Impact of programming language fragmentation on developer productivity," 2010.

[31] Q. Taylor, J. Krein, A. MacLean, and C. Knutson, "An analysis of author contribution patterns in eclipse foundation project source code," *Open Source Systems: Grounding Research*, pp. 269–281, 2011.

[32] J. Casebolt, J. Krein, A. MacLean, C. Knutson, and D. Delorey, "Author entropy vs. file size in the gnome suite of applications," in *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*. IEEE, 2009, pp. 91–94.

[33] F. Khomh, B. Chan, Y. Zou, and A. Hassan, "An entropy evaluation approach for triaging field crashes: A case study of mozilla firefox," in *Reverse Engineering (WCRE), 2011 18th Working Conference on*. IEEE, 2011, pp. 261–270.

[34] M. Cataldo, A. Mockus, J. Roberts, and J. Herbsleb, "Software dependencies, work dependencies, and their impact on failures," *Software Engineering, IEEE Transactions on*, vol. 35, no. 6, pp. 864–878, 2009.

[35] I. Good, "The population frequencies of species and the estimation of population parameters," *Biometrika*, vol. 40, no. 3-4, pp. 237–264, 1953.

[36] J. Camargo, "Revisiting the relation between species diversity and information theory," *Acta biotheoretica*, vol. 56, no. 4, pp. 275–283, 2008.

[37] N. Blüthgen, F. Menzel, and N. Blüthgen, "Measuring specialization in species interaction networks," *BMC ecology*, vol. 6, no. 1, p. 9, 2006.

[38] L. Freeman, "A set of measures of centrality based upon betweenness," *Sociometry*, vol. 40, pp. 35–41, 1977.

[39] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" in *MSR '05: Proceedings of the 2005 international workshop on Mining software repositories*. New York, NY, USA: ACM, 2005, pp. 1–5. [Online]. Available: http://dx.doi.org/10.1145/1083142.1083147

[40] J. Cohen, *Applied multiple regression/correlation analysis for the behavioral sciences*. Lawrence Erlbaum, 2003.

[41] D. Posnett, A. Hindle, and P. Devanbu, "Got issues? do new features and code improvements affect defects?" in *2011 18th Working Conference on Reverse Engineering*. IEEE, 2011, pp. 211–215.