Received: 15 August 2019 | Revised: 24 June 2020 | Accepted: 4 August 2020

DOI: 10.1002/smr.2317

RESEARCH ARTICLE - EMPIRICAL

Journal of Software: Evolution and Process WILEY

Understanding the relation between repeat developer interactions and bug resolution times in large open source ecosystems: A multisystem study

Subhajit Datta[1] | Reshma Roychoudhuri[2] | Subhashis Majumder[2]

[1]Singapore Management University, Singapore

[2]Computer Science and Engineering Department, Heritage Institute of Technology, Kolkata, India

Correspondence
Subhajit Datta, Singapore Management University, 178902 Singapore.
Email: subhajit.datta@acm.org

Abstract

Large-scale software systems are being increasingly built by distributed teams of developers who interact across geographies and time zones. Ensuring smooth knowledge transfer and the percolation of skills within and across such teams remain key challenges for organizations. Towards addressing this challenge, organizations often grapple with questions around whether and how repeat collaborations between members of a team relate to outcomes of important activities. In the context of this paper, the word 'repeat interaction' does not imply a greater number of interactions; it refers to repeat interaction between a pair of developers who have collaborated before. In this paper, we empirically examine such a question using real-world data from three diverse development ecosystems, collectively involving 400,000+ units of work and 600,000+ comments exchanged between numerous developers. Our statistical models consistently establish a counter-intuitive relation between repeat developer interaction and bug resolution times. Our experimental results show that more instances of repeat developer interactions over bug fixing are associated with more time taken for the bugs to be fixed. Given the expanse and variety of the underlying data, our results offer an unexpected set of insights on a key dynamic of collaboration in software development ecosystems. We discuss how these insights can influence the practice of large-scale software development at individual, team and organizational levels.

KEYWORDS

Android, bug resolution, developer interacton, Eclipse, OpenStack, software development ecosystems

1 | INTRODUCTION

1.1 | Background

Civilization, as we know today, runs on software.[1] Our communication, transportation, health care, education, entertainment and other needs are served by software systems in myriad ways. As these systems play increasingly critical roles in our lives—often with serious consequences[2]—development of such systems is also becoming more complex, and with higher stakes. Given the scale and expanse of software systems we frequently use, from operating systems running our mobile devices, to desktop development environments and platforms supporting cloud-based computing, large teams of developers are invariably engaged in their design, development and maintenance. Members

of such teams are mostly spread across geographies and time zones; they seldom, if ever, meet one another face to face, and all members may not ever even be awake at the same point in the circadian cycle.[3] Assembling and governing such teams have engendered a new set of challenges in large-scale software development.[4] These challenges are exacerbated by the phenomenon variously known as *Conway's Law*, *socio-technical congruence* (STC) and *mirroring hypothesis*: the association between the structure of a product and the structure of communication between the individuals building the product.[5-7] As any software practitioner familiar with large development projects knows from experience, and as has also been established by research,[6,8,9] communication dysfunction within teams has detrimental effects on the software being developed.

Factors enabling the building and functioning of truly 'gelled' teams, where the whole is larger than the sum of its parts, have long been of interest to software practitioners.[10] The increasingly global nature of the software development enterprise along with the heightening stakes of software quality leads us to posit that understanding team dynamics is a key element in the practice of software development. In a software development team, developers with various levels of experience and expertise collaborate to fulfil collective responsibilities. Collaboration in such a context rests on the varied interactions between members of the team, as they share knowledge and leverage one another's skills to complete specific activities. Bug resolution has remained one such critical activity ever since software became a major industrial artefact. In the interest of all stakeholders, bugs need to be resolved quickly. Thus, in software systems with large user bases, most intense and consequential developer interactions center around bug fixing.[11-13] This offers an appropriate setting for examining the outcomes of developer interaction.

1.2 | Our research question and its importance

Developers interact across asymmetries in skills, as well as differing levels of contextual familiarity. If we consider a particular bug in a given development ecosystem, at the time of being logged and assigned for resolution, it can be worked upon by a *novice* developer or an *experienced* developer (these terms are formally defined in the context of our study in Section 2.4). As the bug resolution activity progresses over time, novices and experienced developers interact within and across their peer groups, and interactions can repeat between those who have interacted before. Different types of interactions in a team between newcomers and existing members as well as repeat interactions have been studied in diverse other settings such as production of Broadway musicals and scientific collaborations in different disciplines.[14]

Repeat interactions reflect a key dynamic of any collaborative enterprise, and software development is no exception. When we analyse a problem, or discuss or implement a solution or develop a new perspective on a situation, we often reach out to someone with whom we have worked before in the same context.[15] Such repeating interactions seem to present a familiar mechanism for reaching quick resolutions. With this background, we examine the following research question in this study:

In large software development ecosystems, do repeat interactions between developers co-commenting on bugs relate to quicker bug resolution?

In the course of this paper, repeat interaction will refer to interaction between developers who have a prior history of collaboration with each other. Allocation of functionality to different modules has been an enduring concern in software development.[16] Optimal distribution of responsibility among team members is also a critical concern, especially in today's milieu of large-scale distributed development. In a team, experienced developers are able to leverage the 'tribal memory' of the development ecosystem while novices can bring in new points of view.[17] Interactions between two novices, between two experienced, and between a novice and an experienced developer are the different modes of collaboration that are possible in such a setting. Repeated interactions over a specific task, such as resolving a bug, can lead to the establishment of a relationship between the developers working on the task. Such a relationship can be underpinned by a shared understanding of the problem and/or solution spaces, or the complementary nature of skills and experience. Repeat interactions can also be driven by a level of comfort between the interacting individuals in their joint effort towards completion of an activity. Software development methodologies can thus be viewed as an amalgamation of different types of interactions between individuals, which finally influence the outcome of the project.[18] In this context, addressing our research question has implications at the individual, project and collective levels.

Understanding whether and how repeat interactions relate to quicker bug resolution can help individual developers make informed decisions on whom to connect with, for advice and assistance in a large team. For project managers, addressing this question is of importance in several dimensions. It can enable the assignment of collective responsibility to team members who have (or have not!) worked with one another before. Many instances of repeating interactions over a particular unit of work can be a pointer to the complexity or criticality of that unit and trigger a preemptive intervention. Additionally, depending on the insights derived from examining our research question, managers can enable specific types of developer interactions through the use of appropriate tools and processes. For software development with distributed workforces, allocation of work across geographies in the most effective configuration is far from a settled issue, even after two decades of investigations,[19,20] and still continues to the very present.[21] A large-scale empirical examination of the research question can offer actionable insights on whether it is more effective to co-locate developers who repeatedly interact or whether remote collaboration—often logistically more preferable—can suffice.

1.3 | Outline of the paper

In this paper, we address the research question by examining data from three different development ecosystems, offering a variety of operating environments and functionality. As established across disciplines, a *network* offers a useful abstraction for investigating dynamics of interacting individuals.[22] Accordingly, we specify how a developer interaction network (DIN) is constructed for each of the developer ecosystems in Section 2. Construction of the DIN helps us identify different types of interactions of interest to this study. The research question is addressed in terms of examining the influences of these interactions on the resolution time of bugs. Section 3 describes the methodology of our study: we extract a hypothesis from the research question that can be empirically validated, followed by the identification of dependent, independent and control variables to feed into statistical models, development of such models and analysis of the results. In Section 4, we present the results from our models and discuss their implications in the context of our research question. Subsequently, the threats to the validity of our results are identified, along with plans of future work in Section 6. The paper ends with a discussion of related work (Section 7), and summary and conclusions (Section 8).

2 | STUDY SETTING

2.1 | Positioning our case study

According to Runeson and Host, 'Case study is a suitable research methodology for software engineering research since it studies contemporary phenomena in its natural context.'[23] We position the case study reported in this paper by identifying our work with the relevant characteristics of software engineering case studies that Runeson and Host have highlighted.[23] Our case study is *descriptive*and *explanatory* as it portrays a situation of interest in software development ecosystems and seeks to explain the drivers of the situation on the basis of *quantitative* data analysis. Additionally, our study is *positivist* as it 'searches evidence for formal propositions, measures variables, tests hypotheses and draws inferences.'[23] By analysing data from three different software development ecosystems, we use *data (source) triangulation*. Triangulation enables the compensation of measurement or modelling errors and helps enhance the precision of our results.[23]

Robson and McCartan have defined a set of key elements of a research plan.[24] We list the elements below and briefly outline how each element is addressed in this study as follows:

- *Objective: what does the study seek to achieve?* Developer interaction is a key mechanism in the functioning of large and distributed teams as they build and maintain complex software systems. In this study, we seek to achieve an understanding of whether and how repeat interactions between developers relate to the time it takes for bugs to be resolved.
- *The case: what is being studied?* We study instances of repeat interactions between developers in three different software development ecosystems.
- *Theory: what is the frame of reference?* Although developer interactions in large-scale software development have been studied from different perspectives, the very nature of the discipline has precluded the development of an overarching theory.[15] Thus, our study is underpinned by the body of empirical insights that have accumulated over time, some of which have been outlined in Section 7.
- *Research questions: what is being investigated?* We seek to understand whether and how repeat interactions between developers co-commenting on bugs relate to quicker resolution of bugs in large software development ecosystems.
- *Methods: how is the data collected?* We analyse historical datasets from software development repositories (as discussed in detail in Section 2.2) that have been collected, curated and shared for academic research.
- *Selection strategy: where is the data sought?* As mentioned earlier, we have used historical data for this study. Our specific choices of the datasets have been explained in Section 2.2, and the implications of the choices have been discussed in Section 6.

Perry et al. have defined the following criteria for conducting software engineering case studies;[25] along with each criteria, we mention the section number of this paper where that criteria have been addressed:

- *Specification of research questions from the beginning of the study:* Section 1.2.
- *Planned and consistent data collection:* Sections 2.2,2.3 and 2.4.
- *Drawing of inferences from the data to address the research question:* Sections 3 and 4.
- *Offering an explanation, description or causal analysis of the phenomenon being explored:* Section 5.
- *Systematic discussion of threats to validity:* Section 6.

2.2 | Choice of datasets

To empirically examine our research question, we use development data from Android,[*] Eclipse[†] and OpenStack.[‡] Android is a mobile operating system based on a modified version of the Linux kernel and other open source software; it is currently one of the most popular operating systems for handheld devices.[§] Eclipse is widely used as an open source integrated development environment (IDE) for desktop environments with an extensible plug-in system; it is increasingly positioned as a platform to enable collaboration.[¶] OpenStack is an open source platform for cloud computing, enabled for deployment as infrastructure-as-a-service (IaaS).[#]

The choice of our datasets is driven by the intention to examine open source systems from diverse computing paradigms. The open source nature of each of these systems offers a common ground for examining the research question. On the other hand, the systems also represent notable variety in terms of environment (mobile, desktop and cloud) as well as functionality (operating system, IDE and IaaS). Even as we recognize the implications of the choice of our datasets in Section 6, we believe that these systems collectively offer the opportunity for a consummate examination of the research question.

For our study, we used development data as collected, curated and shared by Shihab et al.[26] for Android and by Gonzalez-Barahona et al. for Eclipse and OpenStack data.[27]

2.3 | Pre-processing data

In each of the datasets, bugs were classified in different categories. For our analysis, we only selected those bugs which were resolved and released to ensure that there was closure for all the units of work that are being studied. Since we are examining developer interaction around bugs, we selected only those resolved and released bugs which have been commented upon by three or more unique developers. This filtering criteria helps ensure that there was non-trivial developer interaction around the bugs we are studying.

After pre-processing the datasets, we had 808 bugs with 7,427 comments being exchanged by 2,815 contributors for Android; 13,248 bugs with 227,718 comments being exchanged by 11,326 contributors for Eclipse; and 25,531 bugs with 385,474 comments being exchanged by 4,196 contributors for OpenStack.

For the ease of querying, we persisted all three datasets in specially designed MySQL[‖] databases and developed a Java[**] framework for generating the DINs.

2.4 | Identifying interactions around a bug

In a given development ecosystem, we define a DIN, whose vertices are developers, and two developers are joined by an edge (undirected link) if both have commented on at least one common bug. If this same pair of developers co-comment on another bug again, then no new edge is added to the *DIN*. Instead, the weight of the existing edge between the pair of developers is increased by one for every time they have co-commented. Thus, the weight of any edge of a *DIN* denotes the number of different bugs the developers at the two ends of the edge have both commented upon. In the context of this study, edges of *DIN* that are generated by developers co-commenting on some bug b_i will be denoting *interactions*between developers around b_i. Co-commenting is the instance of three or more developers commenting on a bug. The datasets we studied did not have specific information on the modalities of bug assignment. Thus, even as we cannot specifically determine how bugs were assigned, the processes may have been similar to the ones followed in similar development ecosystems. In large-scale development ecosystems of the types studied here, it is unlikely that all developers know one another personally. Although developer location-related information was not available in the datasets, it is not likely that the developers work face to face, given the size of the developer pool for each dataset. As we focus on the developers (vertices) and their interactions (edges) pertaining to each bug, we first identify each vertex to either represent a *novice* or an *experienced*developer.

A novice is a developer who has not commented on any other bug before the one we are currently considering. A developer who has commented on at least one bug before the one we are currently considering is an experienced developer.[††] On the basis of these definitions, we now define various types of interactions between developers as follows:

*https://www.android.com/
†https://www.eclipse.org/
‡https://www.openstack.org/
§https://en.wikipedia.org/wiki/Android_(operating_system)
¶https://en.wikipedia.org/wiki/Eclipse_(software)
#https://en.wikipedia.org/wiki/OpenStack
‖https://www.mysql.com/
**https://www.java.com/
††Implications of this criterion for identifying Novices and Experienced developer is discussed in Section 6.

- *NewInteraction*: an instance of two novice developers co-commenting on a bug for the first time in a development ecosystem is defined as a NewInteraction. In the DIN corresponding to this ecosystem, an edge between two vertices representing these two novices will indicate a NewInteraction.
- *FreshInteraction*: an instance of one novice developer and one experienced developer co-commenting on a bug in a development ecosystem is defined as a FreshInteraction. In the DIN corresponding to this ecosystem, an edge between a vertex representing the novice and another vertex representing the experienced developer will indicate a FreshInteraction.
- *SeasonedInteraction*: an instance of two experienced developers co-commenting on a bug for the first time in a development ecosystem is defined as a SeasonedInteraction. In the DIN corresponding to this ecosystem, the first edge between vertices representing these two experienced developers will indicate a SeasonedInteraction.
- *RepeatInteraction*: an instance of two experienced developers, who have co-commented on a bug before, co-commenting again on another bug is defined as a RepeatInteraction. In the DIN corresponding to this ecosystem, an edge between vertices representing these two experienced developers will already exist (representing SeasonedInteraction) since they have co-commented on a bug before. For every subsequent interaction between these two experienced developers (i.e., for every RepeatInteraction), the weight of the edge will be increased by 1.

With reference to Figure 1, we will now highlight how these different types of vertices and edges are identified using a simple scenario involving four developers. We start with a temporally ordered list of bugs $L = [b_1, b_2, \ldots, b_i, \ldots, b_n]$ where the ordering is based on the timestamp of the first comment on each bug. In the example scenario in Figure 1, we have assumed that b_i is being commented upon by four developers: d_1, d_2, d_3, d_4. Since all four developers are co-commenting on b_i, there will be $\binom{4}{2} = 6$ edges between them. Let us enumerate the different cases that may arise as follows:

- **Case 1**: top left diagram of Figure 1—all of the developers d_1, d_2, d_3, d_4 have not commented on any bug before they comment on b_i. So, new vertices are generated for all the developers d_1, d_2, d_3, d_4, and they are marked as novices. Finally, all the six edges between these vertices are marked as NewInteractions.
- **Case 2**: bottom left diagram of Figure 1—any one developer, say d_2 has commented on some bug, say b_f before (s)he comments on b_i, while each of d_1, d_3, d_4 posts their first comment on b_i. Evidently, a vertex already exists for d_2, and it is marked as an experienced developer. New vertices are generated for d_1, d_3, d_4, and they are marked as novices. The three edges among d_1, d_3, d_4 are marked as NewInteractions. The other three edges—(d_2, d_1), (d_2, d_3) and (d_2, d_4)—are marked as FreshInteractions.
- **Case 3**: bottom right diagram of Figure 1—among the developers, say d_1 and d_3 have both commented on some bug b_f before they comment on b_i. Evidently, vertices already exist for d_1 and d_3, and they are marked as experienced. Also, an edge already exists between them on account of their co-commenting on b_f. Thus their co-commenting on b_i is marked as a RepeatInteraction, and it is recorded by increasing the weight of the edge (d_1, d_3) by one. New vertices are generated for d_2 and d_4, and they are marked as novices. The single edge (d_2, d_4) is marked as a NewInteraction. The edges (d_1, d_2) and (d_1, d_4) are marked as FreshInteractions. Similarly, the edges (d_3, d_2) and (d_3, d_4) are marked as FreshInteractions.
- **Case 4**: top right diagram of Figure 1—among the developers, say d_1 has commented on bug b_f before (s)he comments on b_i, and d_4 has commented on some other bug b_h before (s)he comments on b_i. Thus, vertices exist for both d_1 and d_4, and they are marked as experienced. However, there is no existing edge between d_1 and d_4, as they have not co-commented on any bug in the past. The edge (d_1, d_4) is added, and it is marked as a SeasonedInteraction. New vertices are added for d_2 and d_3, and they are marked as novices, and an edge (d_2, d_3) is added and marked as NewInteraction. The edges (d_1, d_2), (d_1, d_3), (d_4, d_2), (d_4, d_3) are marked as FreshInteractions.

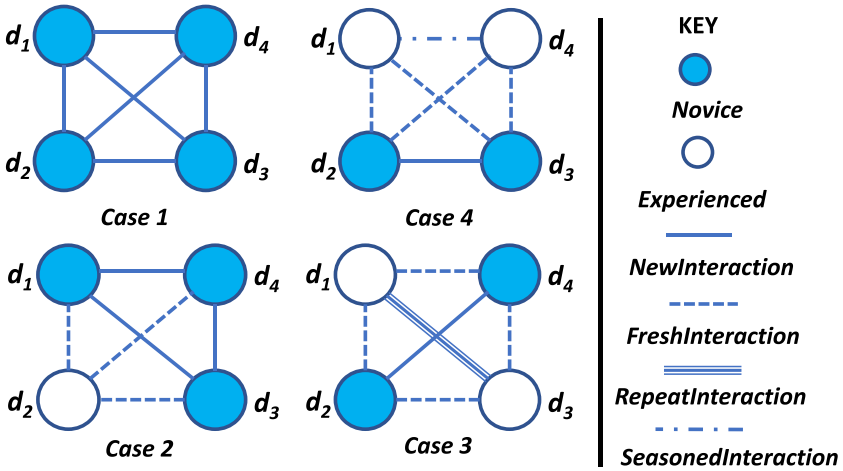

FIGURE 1 Different types of vertices and edges in the Developer Interaction Network pertaining to an example scenario where four developers d_1, d_2, d_3, d_4 comment on a particular bug b_i; diagrams corresponding to Cases 1 to 4 (Section 2.4) are positioned counter-clockwise in the figure above

So in summary, in the context of developers co-commenting on a particular bug, an edge between two novices represent a NewInteraction; an edge between a novice and an experienced developer represent a FreshInteraction; a RepeatInteraction represents an instance of co-commenting on this bug by two experienced developers who have co-commented on some other bug in the past and is captured by increasing the weight of the existing edge by one; and, a SeasonedInteraction represents an edge between two experienced developers who have have not co-commented on some other bug in the past. The above four cases cover all possible configurations of vertices and edges in the example scenario. As we consider each bug in L, the different types of interactions around it are recorded for use in the development of the statistical model in Section 3.3. Across all the bugs in L, the DIN is constructed by considering situations corresponding to the above cases.

3 | METHODOLOGY

Figure 2 outlines the methodology of our study as various elements interact to help define the research question and derive insights from our results. In this figure, the named icons represent the components of the methodology and the numbered arrows denote activities. Existing studies and our understanding of software development ecosystems helped us frame the research question, which was also informed by the extracted data from the particular ecosystems we chose to study. The extracted data in the context of the research question led us to the definition of the DIN. Metrics derived from these networks and other attributes of the extracted data supported the development of statistical models. Results from these models in the general context of existing studies and particular characteristics of the datasets we are studying allowed us to derive insights towards addressing the research question.

In the following subsections, we discuss some of the key aspects of the methodology.

3.1 | Hypothesis definition

To examine the research question introduced in Section 1 using statistical models, we first define a null hypothesis H_0 that can be tested with empirical evidence as follows:

H_0 : *there is no relation between repeat developer interactions and bug resolution times.*

In the context of this study, as presented in Section 1, the corresponding alternate hypothesis H_a is as follows:

H_a : *higher levels of repeat developer interactions relate to shorter bug resolution times.*

We will test H_0 using a regression model that examines the relation between the *dependent variable* – bug resolution time; and the *independent variable* – repeat interactions in the presence of *control variables*. Control variables are factors which are known to influence the outcome in study settings similar to ours.

3.2 | Specification of model variables

We now describe how each model variable is identified on the basis of existing literature and is calculated in the specific context of our study.

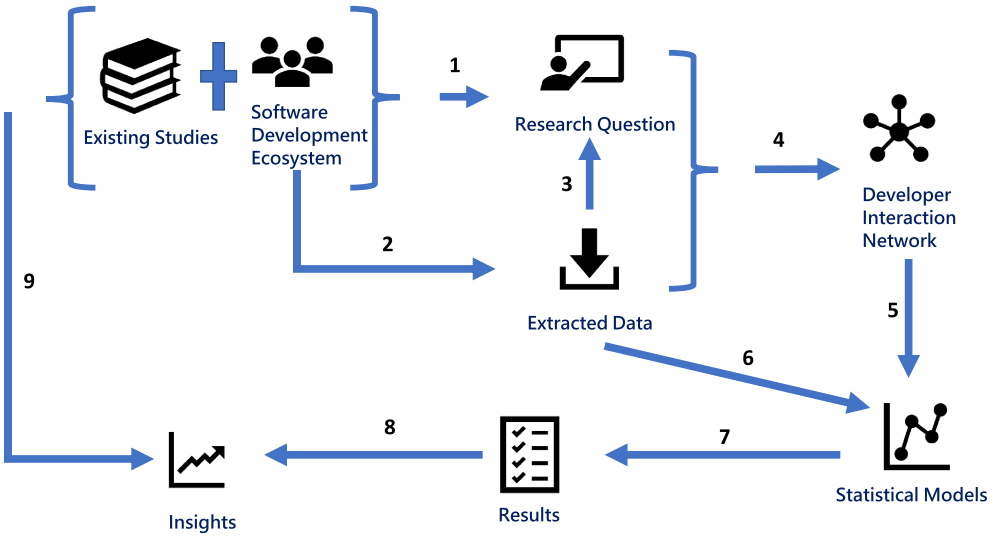

FIGURE 2 Outline of the methodology

3.2.1 | Dependent variable

ElapsedTime: the time it takes for a bug to be resolved is of strong interest to end users and developers and is thus a key outcome of the software development enterprise.[28] In our study setting, the resolution time for a particular bug is calculated by finding the time difference between the first and last activity recorded on the bug. As we have filtered our datasets to only retain resolved and released bugs (as described in Section 2), this time difference is taken to reflect the elapsed time between a bug's opening and closure.

3.2.2 | Independent variable

RepeatInteractions: Section 2.4 elucidates the different types of interactions between the developers in software development ecosystems. As explained in Section 1.2, our research question involves studying the relation between repeat interactions and bug resolution times. The null hypothesis and alternate hypothesis (derived from the research question earlier in this section) specify the examination of one influence of interest on the dependent variable. Thus, we have identified the single independent variable *RepeatInteractions* to include in our models. For a particular bug, this is calculated as the number of edges marked as RepeatInteractions using the method outlined in Section 2.4. This reflects how many times the same pair of developers have repeatedly interacted with each other in the resolution process.

3.2.3 | Control variables

To isolate the relation between the dependent and the independent variable, we consider three categories of control variables—*activity-centric*, *individual-centric* and *interaction-centric*—to account for some of the known influences on software development outcomes, as recognized in existing literature, as follows:

- **Activity-centric control variables** seek to capture factors that relate to the specific activity being worked upon; in this case, bug resolution.
  -*Priority*: in collective enterprises in general, and software development in particular, importance of the task being worked upon by a team is known to affect its outcome.[6,29-32] In our datasets, every bug has an assigned priority. The Android dataset has five levels of priorities ranging from "small" to "blocker"; Eclipse has seven levels of priorities ranging from "trivial" to "blocker", and OpenStack has six ranging from "undecided" to "critical". We coded the priority levels on a numerical scale with 1 being the lowest, each succeeding level denoting higher importance attached to a particular bug.
  -*Interest*: In bug resolution, the number of comments posted on a bug has been found to relate to how quickly it is resolved.[32] In our study, this is calculated as the number of comments posted on a particular bug over its entire life cycle. The number of comments is thus taken as a proxy for the extent of interest among developers around that bug.
  -*Contributors*: The number of developers contributing to the bug resolution process is an important determinant of how quickly the bug gets resolved.[33] In our study setting, commenting on a bug is an essential vehicle for contributing towards its resolution. Thus, this control variable is calculated as the number of unique developers who have commented on a bug.
  -*CommunityInterest*: While each bug has an owner, challenging bugs need wider involvement of the developer community for speedy resolution. The extent to which the development community gets involved in a bug, beyond its immediate owner, has been recognized as an important driver of bug resolution.[9] Accordingly, for a particular bug, we measure this control variable by the number of comments made by developers who *do not* own that bug.
  -*ExternalContributors*: As another dimension of how widely the bug is being attended to by the development community,[9] we calculate this control variable as the number of non-owner developers who have commented on a bug.
- ***Individual-centric control variables*** seek to capture the characteristics of individual developers who own the bugs we are studying.
  -*DeveloperExperience*: The amount of work assigned to a developer has been shown to relate to his/her performance.[6,31,32] To control for this factor in our study, we consider the number of bugs owned by the owner of a particular bug.
  -*OwnerEngagement*: The extent to which a developer is engaged in the overall development ecosystem influences the outcome of his/her individual tasks.[6,29,31,32,34,35] In our study, this control variable is calculated as the number of comments made by the owner of a particular bug across all other bugs in the development ecosystem. This is taken as a proxy for the level of a bug owner's engagement beyond his/her immediate responsibilities.
  -*OwnerSpectrum*: The overall level of familiarity of a developer with the activities (s)he is working on influences his/her work.[29] We calculate this control variable as the number of comments made by the owner of a bug on all bugs owned by him/her. This is taken to reflect the spectrum of the owner's involvement in the development ecosystem.

- ***Interaction-centric control variables*** reflect on the aspects of developer interaction other than the repeat interactions captured by the independent variable. Developer interactions, as reflected in the structure of team communications, are known to affect collective outcomes.[36-39] In our study setting, the following control variables are extracted from the construction of the DIN (Section 2) using the method outlined in Section 2.4:
  -*NewInteractions*: For a particular bug, this is calculated as the number of edges marked as NewInteractions.
  -*FreshInteractions*: For a particular bug, this is calculated as the number of edges marked as FreshInteractions.
  -*SeasonedInteractions*: For a particular bug, this is calculated as the number of edges marked as SeasonedInteractions.

3.3 | Model development

We note that our dependent variable *ElapsedTime* is a continuous variable. Thus, multiple linear regression is chosen as the modelling paradigm, over Poisson or negative Binomial regressions, which are more suited to count variables.[40] Linearity, normality and homoscedasticity of the residuals and the absence of multicollinearity between independent variables are the assumptions underlying multiple linear regression. We verified the residual properties using histograms, Q-Q plots and scatter plots of the standardized residuals. We checked the correlations between the model variables presented in Section 3.2 and in the interest of the most parsimonious models, removed the variables *Interest*, *ExternalContributors* and *Priority* from the final models, as they were strongly correlated with the other variables. The variance inflation factors (VIFs) of the independent variables established the lack of notable multicollinearity in the models (Tables 1 to 9). The Pearson correlation coefficients of the model variables with the dependent variable for each of the datasets are presented in Tables 2, 5 and 8. Figures 3 to 5 present the scatterplots for all pairs of model variables.

As evident from Tables 1, 4 and 7 in each of the datasets, some of the variables are positively skewed. As recommended in such situations, the corresponding variables such as *SeasonedInteractions* and *RepeatInteractions* were transformed in each case by taking second or fourth roots, respectively.[40]

Tables 3, 6 and 9 present results from the multiple linear regression models for Android, Eclipse and OpenStack respectively. The model with the dependent variable and the control variables is shown on the left of each table (Model I), while on the right, the model that additionally

TABLE 1 Android: descriptive statistics of the model variables

Variable	Mean	Standard deviation	Median
ElapsedTime	346.463	281.212	238.93
Contributors	6.244	5.233	4
CommunityInterest	6.574	6.764	4
DeveloperExperience	8,564.033	8,734.472	331
OwnerEngagement	65.405	101.952	1
OwnerSpectrum	24.051	56.42	0
NewInteractions	18.157	47.035	0
FreshInteractions	8.897	15.651	3
SeasonedInteractions	0.691	1.451	0
RepeatInteractions	2.301	1.936	2

TABLE 2 Android: Pearson correlation coefficients of model variables with the dependent variable *ElapsedTime*

Variable	Correlation
Contributors	0.041
CommunityInterest	-0.018
DeveloperExperience	0.27
OwnerEngagement	-0.136
OwnerSpectrum	-0.127
NewInteractions	-0.031
FreshInteractions	0.044
SeasonedInteractions	0.038
RepeatInteractions	0.242

TABLE 3 Android: regression models to understand the influences on *ElapsedTime*

	Model I	Model II
Intercept	**-51.101***	**-122.371****
	0*	0****
	(38.497)	(40.313)
Contributors	96.394*****	83.898*****
	1.794*****	1.561*****
	(10.188)	(10.314)
CommunityInterest	-24.021*****	-18.211*****
	-0.578*****	-0.438*****
	(3.386)	(3.517)
DeveloperExperience	0.012*****	0.012*****
	0.366*****	0.38*****
	(0.001)	(0.001)
OwnerEngagement	0.591****	0.58****
	0.214****	0.21****
	(0.195)	(0.192)
OwnerSpectrum	-0.857****	-0.662***
	-0.172****	-0.133***
	(0.303)	(0.301)
NewInteractions	-5.989 *****	-4.956*****
	-1.002*****	-0.829*****
	(0.677)	(0.696)
FreshInteractions	-4.971*****	-5.69*****
	-0.277*****	-0.317*****
	(1.481)	(1.465)
SeasonedInteractions	-28.327**	-29.445**
	-0.071**	-0.074**
	(15.837)	(15.589)
RepeatInteractions		69.54*****
		0.187*****
		(13.432)
Model parameters		
N	808	808
R^2	0.191	0.218
df	799	798
F	23.619	24.651
Significance level	*****	*****

Note. For each variable, the first row represents the regression coefficient, the second row represents the standardized regression coefficient (in italics) and the third row represents the standard error (in parentheses). *$P \geq 0.1$. **$P \leq 0.1$. ***$P \leq 0.05$. ****$P \leq 0.01$. *****$P \leq 0.001$.

includes the independent variable is presented (Model II). The numbers in italics present the standardized regression coefficients. The standardized regression coefficients give a more precise measure of the extent to which repeat interactions impact elapsed time and quantify how much of the variance of the dependent variable is quantified by the independent and control variables. As specified in the table caption, superscripts of the coefficients indicate the ranges of their respective *P* values. The *P* values are calculated using the *t-statistic*—ratio of each coefficient to its standard error—and the *Student's t-distribution*. The lower portion of each table describes the overall model in terms of the following parameters: *N*, the number of data points in the mode, which in this case, the number of bugs being considered for each dataset; R^2, the coefficient of determination, which is the ratio of the regression sum of squares to the total sum of squares and it indicates the goodness-of-fit of the model; *df*, degrees of freedom; *F*, the Fisher *F*-statistic, which is calculated as the ratio of the variance in the data explained by the linear model divided by the variance unexplained by the model. Each model's *P* value, calculated using the *F*-statistic and the *F*-distribution, establishes whether the

TABLE 4 Eclipse: descriptive statistics of the model variables

Variable	Mean	Standard deviation	Median
ElapsedTime	1.843×10^4	2.156×10^4	9,359.099
Contributors	6.199	5.777	5
CommunityInterest	16.781	11.892	14
DeveloperExperience	2,723.442	4,185.361	955
OwnerEngagement	689.413	1,372.972	109
OwnerSpectrum	678.823	1,368.26	95
NewInteractions	3.206	57.866	0
FreshInteractions	8.788	93.358	0
SeasonedInteractions	6.071	47.554	0
RepeatInteractions	14.73	138.789	6

TABLE 5 Eclipse: Pearson correlation coefficients of model variables with the dependent variable *ElapsedTime*

Variable	Correlation
Contributors	0.212
CommunityInterest	0.173
DeveloperExperience	0.395
OwnerEngagement	-0.121
OwnerSpectrum	-0.12
NewInteractions	0.054
FreshInteractions	0.088
SeasonedInteractions	0.108
RepeatInteractions	0.065

TABLE 6 Eclipse: regression models to understand the influences on *ElapsedTime*

	Model I	Model II
Intercept	**73.271*****	**44.441*****
	0*****	0*****
	(1.278)	(3.153)
Contributors	2.648****	1.932*****
	0.198*****	0.144*****
	(0.23)	(0.24)
CommunityInterest	0.481*****	0.351*****
	0.074*****	0.054*****
	(0.08)	(0.081)
DeveloperExperience	0.007*****	0.006*****
	0.357*****	0.346*****
	(1.457×10^{-4})	(1.467×10^{-4})
OwnerEngagement	0.036*	0.031*
	0.644*	0.558*
	(0.026)	(0.026)
OwnerSpectrum	-0.045**	-0.04*
	-0.792**	-0.712*
	(0.026)	(0.026)

TABLE 6 (Continued)

	Model I	Model II
Intercept	**73.271*****	**44.441*****
NewInteractions	-0.021*	-0.008*
	-0.016*	-0.006*
	(0.026)	(0.026)
FreshInteractions	-0.085*****	-0.072*****
	-0.102*****	-0.087*****
	(0.019)	(0.019)
SeasonedInteractions	1.71*****	2.049*****
	0.048*****	0.057*****
	(0.479)	(0.479)
RepeatInteractions		28.579*****
		0.094*****
		(2.86)
Model parameters		
N	13,248	13,248
R^2	0.198	0.204
df	1.324×10^4	1.324×10^4
F	408.159	376.616
Significance level	*****	*****

Note. For each variable, the first row represents the regression coefficient, the second row represents the standardized regression coefficient (in italics) and the third row represents the standard error (in parentheses). *$P \geq 0.1$. **$P \leq 0.1$. ***$P \leq 0.05$. ****$P \leq 0.01$. *****$P \leq 0.001$.

TABLE 7 Openstack: Descriptive statistics of the model variables

Variable	Mean	Standard deviation	Median
ElapsedTime	1,562.671	2,191.513	859.551
Contributors	4.608	2.076	4
CommunityInterest	11.282	15.688	8
DeveloperExperience	957.485	3,569.569	67
OwnerEngagement	976.865	3,700.769	242
OwnerSpectrum	666.303	3,516.777	96
NewInteractions	0.034	1.008	0
FreshInteractions	0.689	2.685	0
SeasonedInteractions	1.205	4.301	0
RepeatInteractions	8.537	13.595	6

TABLE 8 Openstack: Pearson correlation coefficients of model variables with the dependent variable *ElapsedTime*

Variable	Correlation
Contributors	0.092
CommunityInterest	0.051
DeveloperExperience	0.139
OwnerEngagement	-0.038
OwnerSpectrum	-0.029
NewInteractions	0.005
FreshInteractions	0.05
SeasonedInteractions	0.077
RepeatInteractions	0.052

TABLE 9 Openstack: Regression models to understand the influences on *ElapsedTime*

	Model I	Model II
Intercept	**29.614*****	**26.409*****
	0*****	0*****
	(0.382)	(0.766)
Contributors	0.811*****	-0.066*
	0.08*****	-0.007*
	(0.099)	(0.207)
CommunityInterest	-0.021**	-0.014*
	-0.016*****	-0.01*
	(0.011)	(0.011)
DeveloperExperience	6.879×10^{-4}*****	7.111×10^{-4}*****
	0.117*****	0.12*****
	(3.665×10^{-5})	(3.695×10^{-5})
OwnerEngagement	-0.004*****	-0.004*****
	-0.737*****	-0.748*****
	(3.341×10^{-4})	(3.342×10^{-4})
OwnerSpectrum	0.004*****	0.004*****
	0.695*****	0.705*****
	(3.512×10^{-4})	(3.513×10^{-4})
NewInteractions	-0.14*	-0.054*
	-0.007*	-0.003*
	(0.134)	(0.135)
FreshInteractions	-0.017*	0.117**
	-0.002*	0.015**
	(0.055)	(0.061)
SeasonedInteractions	1.741*****	2.056*****
	0.077*****	0.091*****
	(0.171)	(0.183)
RepeatInteractions		4.395 *****
		0.078*****
		(0.912)
Model parameters		
N	25,531	25,531
R^2	0.043	0.044
df	2.552×10^4	2.552×10^4
F	144.004	130.698
Significance level	*****	*****

Note. For each variable, the first row represents the regression coefficient, the second row represents the standardized regression coefficient (in italics) and the third row represents the standard error (in parentheses). *$P \geq 0.1$. **$P \leq 0.1$. ***$P \leq 0.05$. ****$P \leq 0.01$. *****$P \leq 0.001$.

overall model is statistically significant. If P is less than or equal to the *significance level* for a coefficient of an overall model, we conclude that the corresponding result is statistically significant. We examine the results presented in these tables in Section 4.

4 | RESULTS

With reference to Tables 3, 6 and 9, let us observe the overall model parameters from the lower portions of the tables. Each table corresponds to a dataset as specified in the table caption. In each table, Model I examines the relation between the dependent variable and the control variables, and Model II additionally includes the independent variable. The rows in the top portion of the tables present the details of the model variables

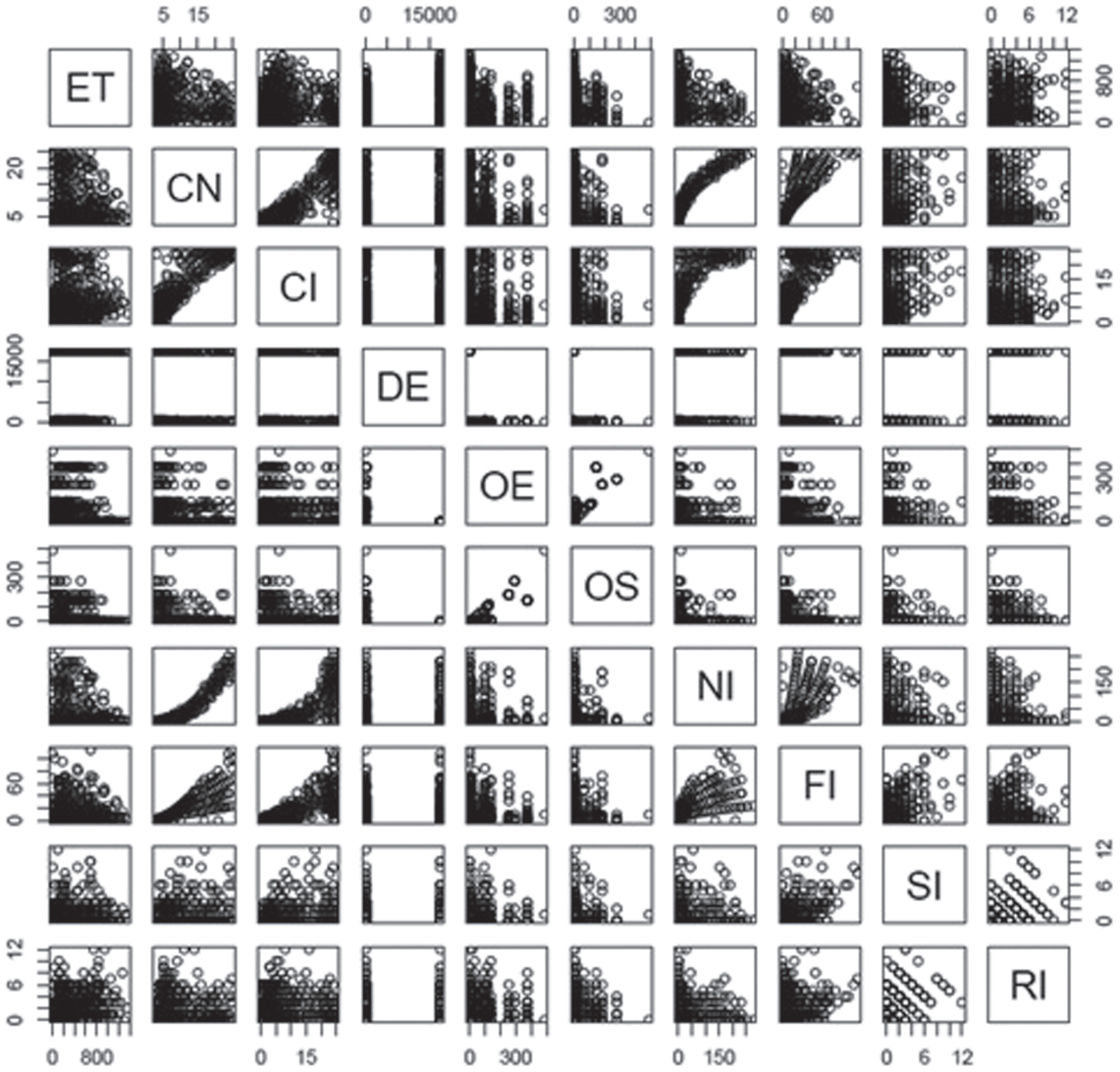

FIGURE 3 Scatterplot for model variables: Android. Abbreviations: ET, *ElapsedTime*; CN, *Contributors*; CI, *CommunityInterest*; DE, *DeveloperExperience*; OE, *OwnerEngagement*; OS, *OwnerSpectrum*; NI, *NewInteractions*; FI, *FreshInteractions*; SI, *SeasonedInteractions*; RI, *RepeatInteractions*

for each of Models I and II. For each variable, the regression coefficient is presented first, followed by the standardized regression coefficient in italics below and by the standard error corresponding to the coefficients in parentheses below. The asterisk(s) or hyphen at the top right corner of each coefficient denotes its respective level of significance, which are specified in the table captions. We note that all models across all three datasets are statistically significant (*significance level* with P value ≤ 0.001 in each case). For Android, Models I and II have goodness-of-fit of 19.1% (R^2 = 0.191) and 21.8% (R^2 = 0.218), respectively. For Eclipse, Models I and II have goodness-of-fit of 19.8% (R^2 = 0.198) and 20.4% (R^2 = 0.204), respectively. And for OpenStack, Models I and II have goodness-of-fit around 4.3% (R^2 = 0.043) and 4.4% (R^2 = 0.044), respectively. On observing Models I and II for each dataset, we see that, in each case, there is an increase in the goodness-of-fit after the addition of the independent variable. Since our models are focused on addressing the research question around repeat developer interactions, we have not included other factors such as size and complexity of the code units required for fixing the bugs, which are likely to increase the overall goodness-of-fit of the models. Let us now focus on how the models help us to test the null hypothesis.

In Model II of each of Tables 3, 6 and 9, we see that the independent variable *RepeatInteractions* has a statistically significant relation with the dependent variable *ElapsedTime*. Thus, in the light of this consistent evidence across all three datasets, we are led to *reject* the null hypothesis H_0 in favour of the alternate hypothesis H_a as defined in Section 3; there is indeed a relationship between repeat interaction between developers and bug resolution times. To understand the directionality of that relationship, we consider the coefficients of *RepeatInteractions* in Model II of Tables 3, 6 and 9: the coefficients for all three datasets are positive (Android: 69.54, P value ≤ 0.001; Eclipse: 28.579, P value ≤ 0.001; OpenStack: 4.395, P value ≤ 0.001) implying a direct relationship between *RepeatInteractions* and *ElapsedTime*.

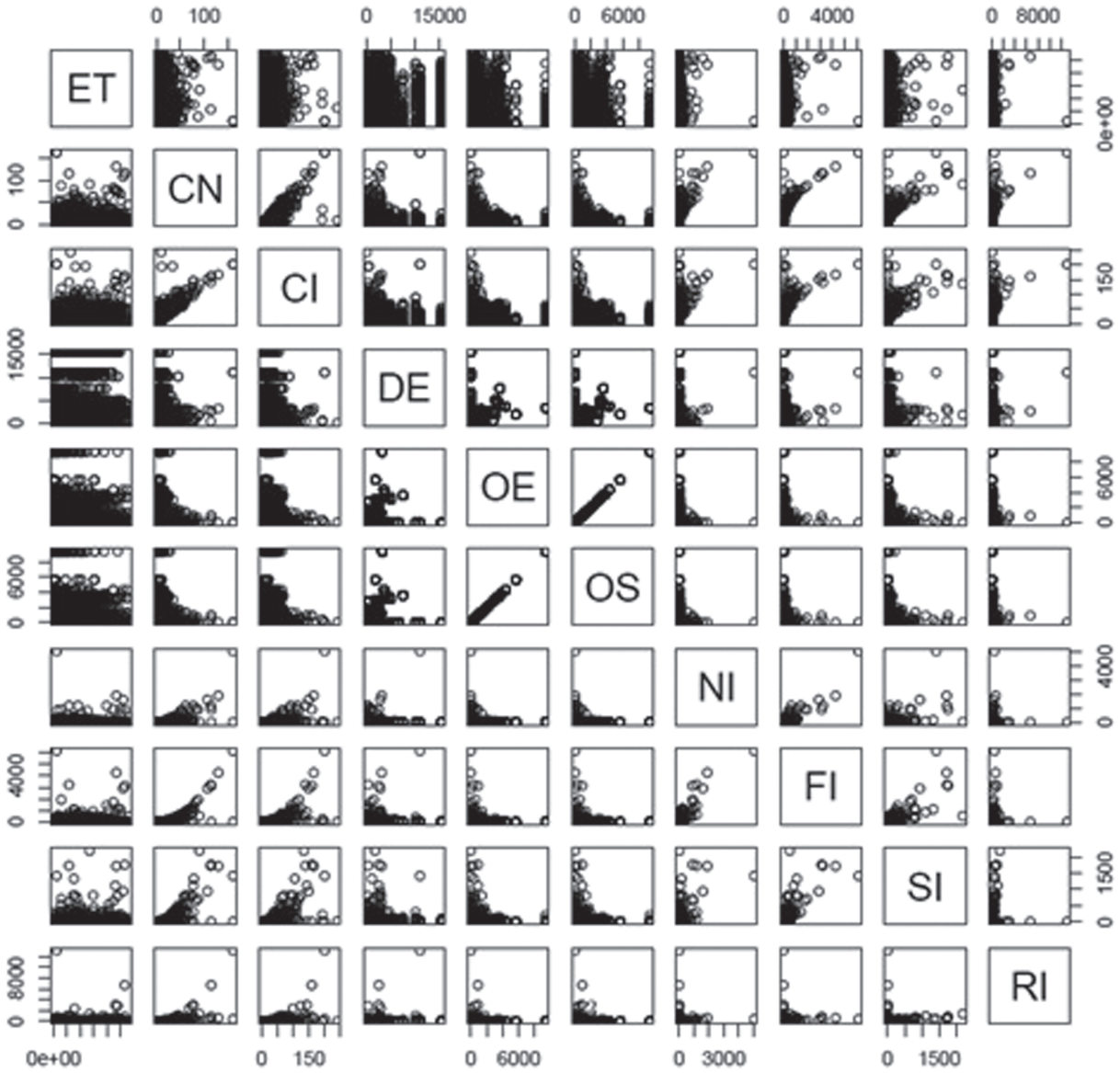

FIGURE 4 Scatterplot for model variables: Eclipse. Abrbeviations: ET, *ElapsedTime*; CN, *Contributors*; CI, *CommunityInterest*; DE, *DeveloperExperience*; OE, *OwnerEngagement*; OS, *OwnerSpectrum*; NI, *NewInteractions*; FI, *FreshInteractions*; SI, *SeasonedInteractions*; RI, *RepeatInteractions*

We also observe the standardized regression coefficients in Tables 3, 6 and 9 for *RepeatInteractions* as 0.187, 0.094 and 0.078 for Android, Eclipse and OpenStack, respectively. Standardized coefficients from a linear regression model represent the parameter estimates that have been obtained when the variances of the model variables have been standardized to be equal to 1. Thus, an increase of one unit in the standard deviation of *RepeatInteractions* will result in an increase of 0.187, 0.094 and 0.078 in the standard deviations of the *ElapsedTime* for Android, Eclipse and OpenStack, respectively. From these standardized regression coefficients, we see that the strength of the influence of *RepeatInteractions* is highest in Android, followed by Eclipse and OpenStack; the strengths of influence of the latter two being approximately half of the former. Among the different types of interactions we examined—*NewInteractions*, *FreshInteractions*, *SeasonedInteractions* and *RepeatInteractions*—the latter most had the highest and second highest strength of influence for Eclipse and OpenStack, respectively. Examining the standardized regression coefficients thus offers us additional insights beyond those obtained from the unstandardized coefficients and *P* values.

In addition to the statistical models a couple of bug instances were analysed to get a better understanding of the behaviour. To ensure that there was no bias in the selection of these bugs, the following procedure was followed: for each dataset, bugs were segregated into four buckets and then segregated by the quartiles of the elapsed times for resolution (corresponding to the dependent variable in our models). So, the first bucket had bugs up to the 25th percentile of elapsed times, the second bucket had bugs from the 26th to the 50th percentile and so on. From each bucket of each dataset, one bug was randomly selected for analysis. This bug selection procedure helped our sample to be representative of

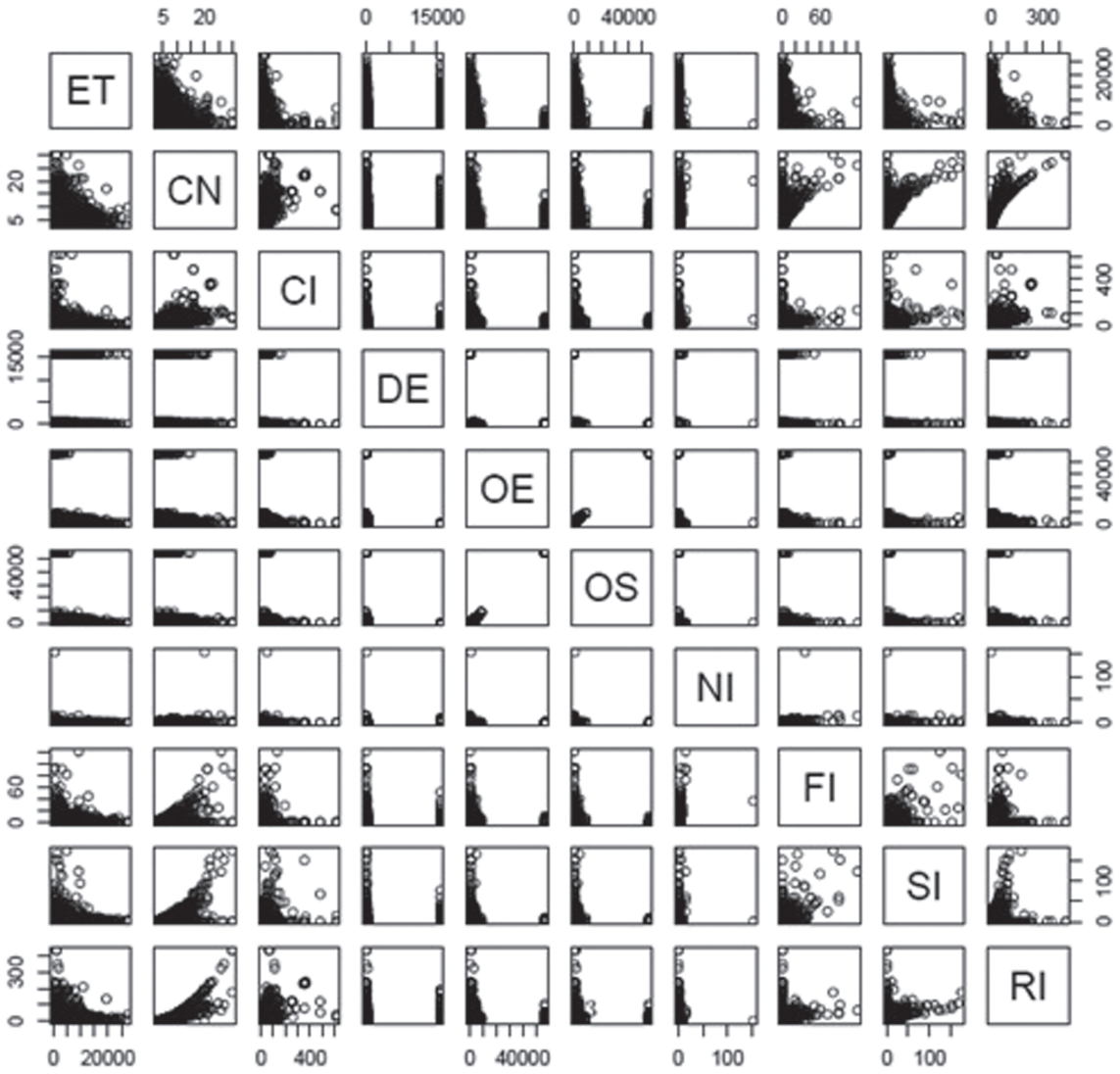

FIGURE 5 Scatterplot for model variables: Openstack. Abbreviations: ET, *ElapsedTime*; CN, *Contributors*; CI, *CommunityInterest*; DE, *DeveloperExperience*; OE, *OwnerEngagement*; OS, *OwnerSpectrum*; NI, *NewInteractions*; FI, *FreshInteractions*; SI, *SeasonedInteractions*; RI, *RepeatInteractions*

a range of bugs by their elapsed times of resolution while free from any selection bias. On the basis of the timeline of comment and values of the model variables on each of the selected bug, the following observations were made: bugs with higher elapsed time of resolution have developer comments over a relatively more extended duration, and the number of repeat interactions of such bugs is higher than other types of interactions. Both of these observations complement with the findings from our statistical models.

Thus, contrary to the situation hypothesized in H_a (Section 3), *we see statistically significant evidence of higher levels of repeat interactions between developers relating to longer bug resolution times, consistently across all three datasets.*

5 | DISCUSSION

Our results present a counter-intuitive perspective of the relationship between repeat developer interactions and bug resolution times. Conventional wisdom seems to suggest that instances of repeat interactions—signifying deeper familiarity, stronger reliance or simply higher co-worker rapport between collaborating developers—should facilitate desirable outcomes for the tasks being worked upon.[14,41,42] However, we find that across the Android, Eclipse and OpenStack development ecosystems, more repeat developer interactions over bug fixing are associated with more time taken for the bugs to be fixed. Before we discuss the practical implications of these results in Section 4, let us try to understand some of the dynamics underlying them.

5.1 | Isolating peripheral influences

As any software practitioner knows, challenging or more visible bugs are more likely to attract higher levels of developer involvement in the resolution process.[12,33,43] As explained in Section 3.2, the activity-centric control variables in our models—*Priority*, *Contributors* and *CommunityInterest*—specifically control for these factors to isolate the effect of the independent variable on the dependent variable. Thus, the relation between developer interaction and bug resolution times that we see in all three development ecosystems can not be attributed to the nature of the bugs.

In the context of this study, interactions are captured between individual developers who have the onus of bug resolution. So, we next consider whether characteristics of individuals may have influenced relation between repeat interaction and bug resolution time. As explained in Section 3.2, the individual-centric control variables in our models—*DeveloperExperience*, *OwnerEngagement* and *OwnerSpectrum*—seek to isolate for effects of various aspects of individual engagement. Thus, the evidence of more repeat interactions relating to slower bug resolution is not influenced by the workload, engagement or spectrum of interest of the developers owning the bugs.

Our research question is concerned with a particular type of developer interactions: repeat ones. In the collaborative space around a bug, interactions are of different types—as captured by the interaction-centric control variables *NewInteractions*, *FreshInteractions* and *SeasonedInteractions*(Section 3.2). As these control variables isolate the effects of all other types of interactions, the association between the independent and dependent variables in our models exclusively reflects the relation between repeat developer interaction and bug resolution time.

As evident form the above discussion, peripheral influences on bug resolution times in our study setting have all been controlled for in the regression models. Thus, our effect of interest is not an artefact of these peripheral factors. Within the limitations presented in Section 6, we now discuss how the effect we consistently see across the datasets can endow a deeper perception of how software developers interact.

5.2 | Towards a deeper understanding of developer interaction

To glean a deeper understanding of developer interaction from the empirical evidence we find of higher levels of repeat interactions between developers relating to longer bug resolution times, we need a perspective of software development vis-a-vis other large-scale human enterprises. Software development is a unique industrial activity, as the 'peopleware' aspect of what is being developed sets it apart from other industrial artefacts.[44] With this background, recent efforts to build a conceptual theory of software development grounded in data from a mixed-methods survey has recognized the importance of improving information exchange in development teams.[15] With the increasing dominance of teams in the production of knowledge across disciplines,[45] it is not surprising that large-scale software development is also becoming increasingly reliant on large teams. There is a growing body of literature on whether and how team assembly and functioning influence outcomes in scientific research.[46,47] However, different ways in which developers interact within software development teams have not been as closely examined yet. Additionally, as mentioned in Section 6, we cannot draw causal inferences from an observational study as the one we report. This leads us to offer some *conjectural explanations* of the relation between repeat interactions and bug resolution times, as evinced in this study.

In the dynamic environment of software production lines serving myriad users, developer interaction remains a critical vehicle for the exchange of information that individuals need to fulfil their responsibilities. Developer interaction is driven by needs around coordination of specific activities, and collaboration towards fulfilment of shared goals. As developers repeatedly interact, they become more familiar with each other's working characteristics and build up a bank of shared experiences around which some techniques and approaches have worked in the past, and some have not. While this familiarity breeds a level of comfort, it can also imperceptibly foster 'groupthink'[48] in the interacting pair. This becomes more detrimental due to a reinforcing feedback mechanism which goads individual developers to interact with those they have interacted with before, on account of being used to one another's ways of functioning. Since all the three development ecosystems we have studied in this paper are open source, it is likely that interactions between developers were governed more by local preferences, rather than management policies common in the development of proprietary systems. So, developers may have cultivated their comfort zones by engaging in repeated interactions.

Engaging with colleagues whom we have never worked with before makes us aware of novel perspectives, complementary skill-sets and untried approaches. On the other hand, repeat interactions can descend into comfortable yet sub-optimal cycles of task completion. Such a dynamic is exacerbated in the defect resolution activities of large and complex systems, where each bug may potentially represent a system flaw with deep ramifications. In the context of this study, this may be reflected in the consistent evidence we see of repeat developer interactions relating to slower bug resolution across all three systems studied. While establishing the causal relationship between this dynamic and our empirical results is beyond the scope of this work, our results can offer key insights for software development practice.

5.3 | Implications for software development practice

Empirical evidence that repeat interactions between developers relate to longer bug resolution times has certain practical implications. To better understand such implications, we need to first examine what repeat interactions may signify. They may be indicative of close monitoring or follow-up on activities between developers. However, assuming that most developers in mature ecosystems do not need such micro-management, repeat interactions are more likely to be a proxy of learning, with more knowledgeable developers offering coaching to others. This perspective can inform individual developers to make conscious choices to work with peers they have not worked before, and carefully culture communication channels that facilitate the circulation of new ideas and approaches. At the team level, this can help identify developers who can serve as effective mentors. While comfort zones in one's immediate working relationships appear desirable, our results point to the pitfalls of being too ensconced in them. By offering a counter-intuitive perspective on repeat interactions, the insights from this study can help facilitate more effective collaboration. Addressing repeat interactions in the structuring and governance of teams presents an interesting dichotomy. While repeat interactions may point to instances of individuals getting along well together and thus fostering overall team harmony, we see evidence of some of its insidious fallouts. So, our results can enable project managers to develop an evidence-based approach towards balancing familiarity and novelty in intra-team interactions. Software development remains a highly people-centric activity.[44] Even if team size grows linearly, connections between team grow quadratically, with concomitant effects that have been long recognized.[49] So, by specifically encouraging other types of interactions over repeat ones—on the basis of our results—organizations can have a way of addressing the combinatorial complexity of team communications.

6 | THREATS TO VALIDITY AND FUTURE WORK

In this paper, we report results from an observational study across three software development ecosystems. As this is not a controlled experiment, correlations observed from our statistical models do not necessarily imply causation. However it needs to be underscored that in a study setting such as ours, controlled experiments are very difficult—if at all possible—to design and execute. Thus, observational studies in such situations can yield useful insights.[50]

Construct validity is concerned with correct measurements of variables. As discussed in Sections 2 and 3, the model variables are calculated either by querying the datasets or from the DIN as generated for each system. The *DINs* have been constructed from the instances of co-commenting between developers; this network construction protocol is consistent with the ones adopted in similar studies.[9] Thus, we do not see notable threats to construct validity.

Internal validity seeks to ensure that a study is free from systematic errors and biases. We have used historical data available in the public domain that has been collected and shared for research purposes[26,27] in this study. Thus, we do not expect *mortality* (instances of subjects withdrawing from a study when the data are being collected) or *maturation* (characteristics of subjects changing during the course of the study, outside the purview of the research parameters) to pose notable threats. As explained in Section 3, we use a consistent preprocessing criteria for each of the datasets. This consistency allows us to arrive at comparable findings across the datasets, even as the number of data points after filtering varies across datasets, with Android having fewer bugs compared with Eclipse and OpenStack. However, since we have relied on curated data as made available by Shihab et al.[26] and Gonzalez-Barahona et al.,[27] errors and omissions if any, introduced during parsing the raw data, can pose threats to internal validity. We believe this threat to be minimal, as the papers presenting the curated data were peer-reviewed before publication.

External validity is about establishing the generalizability of the results from a study. We have discussed the choice of our datasets in Section 2. The choice of the particular datasets for this study were based on considerations of availability of the datasets from reliable, peer-reviewed sources[26,27] and the suitability of the datasets for addressing our research question in terms of the availability of the elements of the datasets needed to consistently compute all the model variables. As evident from the discussions in previous studies,[26,27,51] collecting and curating data from large-scale software development ecosystems is an enterprise requiring significant time and effort. Such an enterprise was beyond the scope of this study. Implications of the fact that we did not collect the data ourselves have been recognized in the preceding discussion of the threats to internal validity. While we believe our choice covers a range of software development ecosystems, we do not claim to have a representative sample. As indicated by the standardized regression coefficients, the size of influence of the independent variable on the dependent variable ranges from 0.187 to 0.078 on a scale of 1. The size of these influences, along with factors identified above, indicate that our results are not generalizable as yet. However, in the study of software development ecosystems, enduring insights have been arrived at, from studies of limited number of systems.[30,37] So, despite the lack of generalizability, our results can offer a useful perspective on the research question.

A study's *reliability* is established when its results can be replicated. Given access to the datasets, our results can be fully replicated. Towards that end, we share a replication package.[‡‡]

[‡‡]https://bitbucket.org/subhajit_datta/heritage-reshma-2017

A study such as ours rests on a set of assumptions. Let us examine the *implications of our assumptions*. As mentioned in Section 2.3, we selected only those bugs which were commented upon by three or more developers. The objective of this study is to examine how repeat developer interactions relate to bug resolution times. A bug which has had a single commenter or just two commenters does not offer an appropriate context for triadic closure[22] among the developers co-commenting on it. Thus, bugs with three or more commenters have been chosen to aptly reflect developer interaction around them. We have distinguished an experienced developer from a novice on the basis of the former having commented on at least one bug in the past. This distinction is based on the assumption that even commenting on one bug—vis-a-vis none—endows a developer with a general sense of the development ecosystem. As there is no established criteria for identifying experienced developers versus novices in a study setting such as ours, we followed a protocol similar to the one used for distinguishing 'newcomers' and 'incumbents' across a variety of collaborative enterprises, as reported by Guimera et al.[14] We recognize that a developer who commits patches related to bugs but does not comment on them will not be considered an experienced developer by our criteria, and this can represent a threat to the validity of our results. However, in the highly interactive culture of the types of software development ecosystems we have examined in this study, it is quite unlikely for a developer not to participate in commenting at all. Thus, the extent of this threat is not expected to be large. We assumed that the priorities of the bugs were not changed during their resolution process. This assumption was based on the lack of such information in our data sources.[26,27] As mentioned in Section 3.2.1, the resolution time for a bug was calculated as the time difference between the first and last activities recorded on that bug. This construction is based on the assumption of a linear flow in the bug resolution process such as open-investigate-close. Our datasets did not provide information on whether some bugs were reopened after closure. Thus, if a particular bug was closed and then reopened, the resolution time subsumes these changes of status.

In our future work, we plan to examine the reasons behind the counter-intuitive results (Section 4) in greater detail. Co-commenting is one aspect of developer interaction. Other aspects such as co-committing of code units,or collaborative review activities can offer interesting settings for studying the relationship between repeat developer interaction and outcomes. Replicating our results across other development ecosystems will also help to address the threat to external validity. Additionally, we plan to work towards extending the empirical evidence into a theory of developer interaction in large-scale software development.

7 | RELATED WORK

We have already referred to some existing studies while positioning our work, introducing the model variables and discussing the results in preceding sections. In this section, we briefly outline some other related work.

7.1 | Team assembly and functioning

Fully understanding the role of communications in collaborative work presents enduring challenges. While science and technology is conventionally believed to have been dominated by individual brilliance, the role of teams have been increasingly getting research attention,[45] with focus on the transition from individual to collaborative working profiles.[52] With a recognition of the immanence of team work in nearly all aspects of our lives today,[53] the role of teams in industrial software development have also come to be examined closely.[54]

As teams began to be considered as the building blocks of modern organizational structure,[55] efforts towards understanding team assembly[56] and optimal team functioning[57] became areas of study. Different models have been suggested on how to develop a team and to gauge the suitability of one team structure over others, particularly in operational contexts.[55]

Effective team assembly involves concerns such as defining the roles and responsibilities of team members, setting up communication channels between them and deciding on performance goals for every team member. Towards these ends, conscious efforts to develop confidence and interpersonal skills of the team members have been shown to have a positive effect on the team performance.[56] It has also been seen that various factors such as wage, motivation and training affected the productivity and effectiveness of teams.[58] Selection of individuals for team membership presents interesting challenges,[59] with results showing that people usually tend to select those who are expected to have a positive effect on the team's functioning.[60]

Several studies have acknowledged the importance of diversity in skill and experience in team composition.[61] The effects of such diversity differ on the basis of specific needs of the team. For example, a project that needed highly knowledgeable workers functioned well with a team that had members of diverse educational background and age; however, these were not significant determinants of project success in other situations.[61] Different models have also been proposed to measure the effectiveness of a team's functioning.[62]

In a distributed development environment like software development, communication and collaboration among developers play an important role with far reaching impact on the outcome of a project.[63] While every phase of software development demands such collaboration, one of the important phases of such collaboration is bug fixing.[13] Optimizing response times on bug fixing is an area of active research.[64,65] Defect resolution thus also provides an ideal backdrop against which different types of developer collaborations and interactions can be effectively studied.

What makes collaborations succeed has been the subject of several investigations. It has been seen that prior acquaintance between workers seemed to bring them closer, irrespective of the actual geographical separation, along with positive effect that a history of prior collaboration seems to have on current collaborations.[66,67] It was also seen that scientific researchers with a body of effective past collaborations had greater chance of securing promising opportunities for future collaborations,[68] and collaborations contribute to the performance of an individual worker.[69] Our results complement these existing studies by offering a new set of insights on the role of repeat developer interactions in large-scale software development.

7.2 | Relationship between interaction and team outcomes in software development

Understanding the genesis of defects and the factors influencing their resolution have been of interest as software systems have become more complex over time.[42,70,71] Ever since software development started becoming a global enterprise at the turn of the 21st century, development teams came to be increasingly distributed, both geographically and temporally.[19,72] However, when the speed of task completion is considered, there is evidence that distributed teams take more time to complete their work items than co-located as reported by Herbsleb and Mockus.[30] From a study of open source projects, Grewal et al. have found that network embeddedness relates to project success.[39] With concomitant increase in the size and diversity of such teams, researchers focused on a deeper understanding of how interaction between team members relates to collective outcomes from the team, such as quality of the team's work products, as manifested in defects, and the speed of task completion. By studying the relationship between distributed development and software quality in a case study of Windows Vista, Bird et al. reported a negligible difference in the defects in code components after they were released, between those that were developed by distributed versus co-located teams.[73] As reported by de Souza and Redmiles, developers in a team use various strategies to mitigate the effects of dependencies between their shared deliverables.[74] Meneely et al. found team size and its linear growth to be correlated with subsequent product quality in a longitudinal study.[38]

The alignment of communication structure and the structure of the system that is being built by the group of communicating individuals has been canonized as *Conway's Law*.[5] Cataldo et al. captured the spirit of Conway's Law in the software development context as STC to investigate the effects of dependencies in software development.[6] The authors found that congruence of developers' coordination characteristics with their needs for coordinating with other developers led to quicker resolution of change requests. Cataldo et al. have also studied how various dependency measures – syntactic, logical and workflow—perform with respect to one another in their relationship to defects;[75] logical dependencies were found to explain most of the variance, followed by syntactic dependencies and workflow. A study by Cataldo and Herbsleb featuring two large projects found that lack of alignment between coordination requirements and actual coordination activities related to higher occurrence of defects.[76] Wolf et al. have studied how successful coordination outcome and communication structures relate to one another building a predictive model that includes several communication structure metrics.[37] The idea of STC has been extended by Wagstrom et al. to distinguish between developer communication that is general in nature, versus communication around task dependencies;[32] the former is found to have no notable benefit, while the latter relates to quicker resolution of defects. A large-scale study of developer interaction in the development ecosystem of a major industrial product has revealed statistically significant effects of connection and clustering on the number of defects in the teams' work products, even after controlling for the effects of work item dependency, system age, developer expertise and experience, geographic dispersion, STC and the number of files changed.[4]

Our study complements these existing results by investigating an aspect of developer interaction that has not received notable attention so far in the context of large-scale software development. To the best of our knowledge, this is among the earliest studies of repeat developer interaction and its association with bug resolution times. Additionally, unlike some of the studies referred to earlier, our results are corroborated by replication across three large-scale systems.

8 | SUMMARY AND CONCLUSIONS

In this paper, we have empirically examined a research question on the relation between repeat developer interactions and bug resolution times in large-scale software development. We have replicated our results on real-world development data from three systems—Android, Eclipse and OpenStack—each representing a distinct computing environment and line of functionality. Having extracted networks of developer interactions from instances of co-commenting on bugs, we identify repeat interactions between developers around a particular bug from the network topology. Examining the relation between repeat developer interactions and bug resolution times using statistical models leads us to counter-intuitive results. Belying expectations that more repeat interactions—representing an enhanced level of familiarity and virtuosity—will be associated with more desirable outcomes, we find evidence to the contrary. *Higher levels of repeat collaborations between developers are found to be related to longer bug resolution times.* The results are statistically significant, consistent across all three datasets and hold after controlling for other effects relevant to the characteristics of bugs, individuals and interactions. Our results offer actionable insights on the practice of large-scale software

development: they can help individual developers make informed choices on their interaction with peers, guide project managers in adequately balancing important concerns in team assembly and governance and facilitate organizations in the effective set-up and sustenance of large-scale software ecosystems.

ORCID

Subhajit Datta https://orcid.org/0000-0001-9161-7951

REFERENCES

1. Stroustrup B. The problem with programming. https://www.technologyreview.com/InfoTech/17987/?a=f
2. Gelles D. Boeing fix for 737 max software is delayed. https://www.nytimes.com/2019/04/01/business/boeing-737-max-software-fix.html#:~:text=Boeing's%20software%20update%20for%20its,to%20the%20Federal%20Aviation%20Administration.&text=On%20Monday%2C%20the%20F.A.A.%20made,timing%20had%20been%20pushed%20back
3. Wagstrom P, Datta S. Does latitude hurt while longitude kills? Geographical and temporal separation in a large scale software development project. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014. New York, NY, USA: ACM; 2014:199-210. https://doi.acm.org.library.sutd.edu.sg:2048/10.1145/2568225.2568279
4. Datta S. How does developer interaction relate to software quality? An examination of product development data. *Empirical Soft Eng*. 2018;23(3):1153-1187. https://doi.org/10.1007/s10664-017-9534-0
5. Conway M. How do committees invent? *Datamation J*. 1968:28-31.
6. Cataldo M, Herbsleb JD, Carley KM. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. Kaiserslautern, Germany: ACM; 2008:2-11. https://portal.acm.org/citation.cfm?id=1414008
7. Kwan I, Cataldo M, Damian D. Conway's law revisited: the evidence for a task-based perspective. *IEEE Software*. 2012;29(1):90-93.
8. Ehrlich K, Cataldo M. All-for-one and one-for-all?: a multi-level analysis of communication patterns and individual performance in geographically distributed software development. In: Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, CSCW '12. New York, NY, USA: ACM; 2012:945-954. https://doi.acm.org/10.1145/2145204.2145345
9. Datta S, Sindhgatta R, Sengupta B. Talk versus work: characteristics of developer collaboration on the jazz platform. In: Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications, OOPSLA '12. New York, NY, USA: ACM; 2012:655-668. https://doi.acm.org/10.1145/2384616.2384664
10. Humphrey WS. Introduction to the team software process. SEI Series in Software Engineering; 1999.
11. Weib C, Premraj R, Zimmermann T, Zeller A. How long will it take to fix this bug? In: MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories. Washington, DC, USA: IEEE Computer Society; 2007:1.
12. Guo PJ, Zimmermann T, Nagappan N, Murphy B. Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE '10. New York, NY, USA: ACM; 2010:495-504. https://doi.acm.org/10.1145/1806799.1806871, event-place: Cape Town, South Africa.
13. Wang S, Zhang W, Yang Y, Wang Q. DevNet: exploring developer collaboration in heterogeneous networks of bug repositories. In: 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement IEEE; 2013:193-202.
14. Guimera R, Uzzi B, Spiro J, Amaral LAN. Team assembly mechanisms determine collaboration network structure and team performance. *Science (New York NY)*. 2005;308(5722):697-702. https://www.ncbi.nlm.nih.gov/pubmed/15860629, PMID: 15860629.
15. Baltes S, Diehl S. Towards a theory of software development expertise. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2018. New York, NY, USA: ACM; 2018:187-200. https://doi.acm.org/10.1145/3236024.3236061
16. Parnas DL. On the criteria to be used in decomposing systems into modules. *Commun ACM*. 1972;15(12):1053-1058.
17. Booch G. Tribal memory. *IEEE Software*. 2008;25(2):16-17.
18. Sajeev ASM, Datta S. Introducing programmers to pair programming: a controlled experiment. In: Baumeister H, Weber B, eds. *Agile Processes in Software Engineering and Extreme Programming*, Lecture Notes in Business Information Processing, vol. 149: Springer Berlin Heidelberg; 2013:31-45. https://doi.org/10.1007/978-3-642-38314-4_3
19. Olson GM, Olson JS. Distance matters. *Hum-Comput Interact*. 2000;15(2):139-178. https://doi.org/10.1207/S15327051HCI1523_4
20. Warshaw J, Whittaker S, Matthews T, Smith BA. When distance doesn't really matter: effects of geographic dispersion on participation in online enterprise communities. In: Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing, CSCW '16. New York, NY, USA: ACM; 2016:335-345. https://doi.acm.org/10.1145/2818048.2835237
21. Gilrane V. Working together when we are not together. https://www.blog.google/inside-google/working-google/working-together-when-were-not-together/2019.
22. Newman MEJ. The structure and function of complex networks. Cond-Mat/0303516 (Mar. 2003). *SIAM Review*. 2003;45:167-256.
23. Runeson P, Höst M. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Eng*. 2008;14(2):131. https://doi.org/10.1007/s10664-008-9102-8
24. Robson C, McCartan K. *Real World Research*. 4th edition. Hoboken: John Wiley & Sons; 2015.
25. Perry DE, Sim SE, Easterbrook S. Case studies for software engineers. In: 29th Annual IEEE/NASA Software Engineering Workshop - Tutorial Notes (SEW'05); 2005:96-159.
26. Shihab E, Kamei Y, Bhattacharya P. Mining challenge 2012: The android platform. In: 2012 9th IEEE Working Conference on Mining Software Repositories (MSR); 2012:112-115.
27. Gonzalez-Barahona JM, Robles G, Izquierdo-Cortazar D. The metricsgrimoire database collection. In: Proceedings of the 12th Working Conference on Mining Software Repositories, MSR '15. Piscataway, NJ, USA: IEEE Press; 2015:478-481. https://dl.acm.org/citation.cfm?id=2820518.2820592

28. Datta S, Lade P. Will this be quick?: a case study of bug resolution times across industrial projects. In: Proceedings of the 8th India Software Engineering Conference, ISEC '15. New York, NY, USA: ACM; 2015:20-29. https://doi.acm.org.libproxy.smu.edu.sg/10.1145/2723742.2723744
29. Espinosa JA, Slaughter SA, Kraut RE, Herbsleb JD. Familiarity, complexity, and team performance in geographically distributed software development. *Org Sci*. 2007;18(4):613-630. https://orgsci.journal.informs.org/content/18/4/613
30. Herbsleb JD, Mockus A. An empirical study of speed and communication in globally distributed software development. *IEEE Trans Softw Eng*. 2003;29:481-494. https://portal.acm.org/citation.cfm?id=1435631.859041
31. Cataldo M, Wagstrom PA, Herbsleb JD, Carley KM. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In: Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work, CSCW '06. New York, NY, USA: ACM; 2006:353-362. https://doi.acm.org/10.1145/1180875.1180929
32. Wagstrom P, Herbsleb J, Carley K. Communication, team performance, and the individual: bridging technical dependencies. Academy of Management Conference; 2010.
33. Datta S, Sarkar P, Das S, Sreshtha S, Lade P, Majumder S. How many eyeballs does a bug need? an empirical validation of Linus' law. In: Cantone G, Marchesi M, eds., Lecture Notes in Business Information Processing, vol. 179: Springer International Publishing; 2014:242-250. https://doi.org/10.1007/978-3-319-06862-6_17
34. Faraj S, Sproull L. Coordinating expertise in software development teams. *Manage Sci*. 2000;46(12):1554-1568.
35. Fong Boh W, Slaughter SA, Espinosa JA. Learning from experience in software development: a multilevel analysis. *Manage Sci*. 2007;53(8):1315-1331. https://doi.org/10.1287/mnsc.1060.0687
36. Curtis B, Krasner H, Iscoe N. A field study of the software design process for large systems. *Commu ACM*. 1988;31:1268-1287.
37. Wolf T, Schroter A, Damian D, Nguyen T. Predicting build failures using social network analysis on developer communication. In: Proceedings of the 31st International Conference on Software Engineering. IEEE Computer Society; 2009:1-11. https://portal.acm.org/citation.cfm?id=1555017
38. Meneely A, Rotella P, Williams L. Does adding manpower also affect quality?: an empirical, longitudinal analysis. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE '11. New York, NY, USA: ACM; 2011:81-90.
39. Grewal R, Lilien GL, Mallapragada G. Location, location, location: how network embeddedness affects project success in open source systems. *Manage Sci*. 2006;52(7):1043-1056.
40. Tabachnick BG, Fidell LS. *Using Multivariate Statistics*. Boston: Pearson Education; 2007.
41. Galbraith JK. *The Affluent Society* 1st ed.. Boston, Massachusetts, United States: Houghton Mifflin; 1998.
42. Jeong G, Kim S, Zimmermann T. Improving bug triage with bug tossing graphs. In: Esec/fse '09: Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM Sigsoft Symposium on the Foundations of Software Engineering. New York, NY, USA: ACM; 2009:111-120.
43. Asaduzzaman M, Bullock MC, Roy CK, Schneider KA. Bug introducing changes: a case study with android. In: Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on; 2012:116-119.
44. DeMarco T, Lister T. *Peopleware: Productive Projects and Teams*. Boston, Massachusetts, United States: Addison-Wesley; 1987.
45. Wuchty S, Jones BF, Uzzi B. The increasing dominance of teams in production of knowledge. *Science*. 2007;316(5827):1036-1039.
46. Wu L, Wang D, Evans JA. Large teams develop and small teams disrupt science and technology. *Nature*. 2019;566(7744):378. https://www.nature.com/articles/s41586-019-0941-9
47. Milojevic S. Principles of scientific research team formation and evolution. *Proc Nat Acad Sci United States of America*. 2014;111(11):3984-3989.
48. Turner M, Pratkanis A. Twenty-five years of groupthink theory and research: lessons from the evaluation of a theory. *Org Behav Human Dec Proc*. 1998;73(2):105-115.
49. Brooks FP. *The Mythical Man-Month: Essays on Software Engineering*, 20th anniversary edition. Boston, Massachusetts, United States: Addison-Wesley; 1995.
50. Shadish WR, Cook TD, Campbell DT. *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. 2edition. Boston: Wadsworth Publishing; 2001.
51. Hamasaki K, Kula RG, Yoshida N, Cruz AEC, Fujiwara K, Iida H. Who does what during a code review? Datasets of OSS peer review repositories. In: Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13. Piscataway, NJ, USA: IEEE Press; 2013:49-52. https://dl.acm.org/citation.cfm?id=2487085.2487096
52. Whitfield J. Collaboration: group theory. *Nature News*. 2008;455(7214):720-723.
53. Kozlowski SW, Ilgen DR. Enhancing the effectiveness of work groups and teams. *Psych Science Public Int*. 2006;7(3):77-124.
54. DeSanctis G, Jackson BM. Coordination of information technology management: team based structures and computer based communication systems. *J Manage Info Syst*. 1994;10(4):85-110.
55. Mathieu JE, Tannenbaum SI, Donsbach JS, Alliger GM. A review and integration of team composition models: moving toward a dynamic and temporal framework. *J Manage*. 2014;40(1):130-160.
56. Klein C, DiazGranados D, Salas E, et al. Does team building work? *Small Group Res*. 2009;40(2):181-222.
57. Bercovitz J, Feldman M. The mechanisms of collaboration in inventive teams: composition, social networks, and geography. *Res Policy*. 2011;40(1):81-93.
58. Tohidi H. Teamwork productivity & effectiveness in an organization base on rewards, leadership, training, goals, wage, size, motivation, measurement and information technology. *Procedia Com Sci*. 2011;3:1137-1146.
59. Millhiser WP, Coen CA, Solow D. Understanding the role of worker interdependence in team selection. *Org Sci*. 2011;22(3):772-787.
60. Hinds PJ, Carley KM, Krackhardt D, Wholey D. Choosing work group members: balancing similarity, competence, and familiarity. *Org Behav Dec Proc*. 2000;81(2):226-251.
61. Kearney E, Gebert D, Voelpel SC. When and how diversity benefits teams: the importance of team members' need for cognition. *Acad Manage J*. 2009;52(3):581-598.
62. Tohidi H, Tarokh MJ. Productivity outcomes of teamwork as an effect of information technology and team size. *Int J Prod Econ*. 2006;103(2):610-615.
63. Bruegge B, Dutoit AH, Wolf T. Sysiphus: enabling informal collaboration in global software development. In: 2006 IEEE International Conference on Global Software Engineering (ICGSE'06) IEEE; 2006:139-148.

64. Zou W, Xia X, Zhang W, Chen Z, Lo D. An empirical study of bug fixing rate. In: 2015 IEEE 39th Annual Computer Software and Applications Conference, Vol. 2 IEEE; 2015:254-263.
65. Zhang F, Khomh F, Zou Y, Hassan AE. An empirical study on factors impacting bug fixing time. In: 2012 19th Working Conference on Reverse Engineering IEEE; 2012:225-234.
66. Dahlander L, McFarland DA. Ties that last: tie formation and persistence in research collaborations over time. *Admin Sci Quart*. 2013;58(1):69-110.
67. Fleming L, Mingo S, Chen D. Collaborative brokerage, generative creativity, and creative success. *Admin Sci Quart*. 2007;52(3):443-475.
68. Newman MarkEJ. Clustering and preferential attachment in growing networks. *Phys Rev E*. 2001;64(2):25102.
69. Uzzi B, Spiro J. Collaboration and creativity: the small world problem. *American J Socio*. 2005;111(2):447-504.
70. Koru AG, Liu H. Building defect prediction models in practice. *IEEE Softw*. 2005;22(6):23-29. https://doi.org/10.1109/MS.2005.149
71. Zimmermann T, Nagappan N. Predicting defects with program dependencies. In: Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on; 2009:435-438.
72. Olson GM, Olson JS. Mitigating the effects of distance on collaborative intellectual work. *Econ Innova New Tech*. 2002:27-42.
73. Bird C, Nagappan N, Devanbu P, Gall H, Murphy B. Does distributed development affect software quality? An empirical case study of windows vista. In: Proceedings of the 31st International Conference on Software Engineering, ICSE '09. Washington, DC, USA: IEEE Computer Society; 2009:518-528.
74. de Souza CRB, Redmiles DF. An empirical study of software developers' management of dependencies and changes. In: Proceedings of the 30th international conference on software engineering, ICSE '08. New York, NY, USA: ACM; 2008:241-250.
75. Cataldo M, Mockus A, Roberts JA, Herbsleb JD. Software dependencies, work dependencies, and their impact on failures. *IEEE Trans Softw Eng*. 2009;35(6):864-878.
76. Cataldo M, Herbsleb JD. Coordination breakdowns and their impact on development productivity and software failures. *IEEE Trans Soft Eng*. 2013;39(3):343-360.

How to cite this article: Datta S, Roychoudhuri R, Majumder S. Understanding the relation between repeat developer interactions and bug resolution times in large open source ecosystems: A multisystem study. *J Softw Evol Proc*. 2020;e2317. https://doi.org/10.1002/smr.2317