How Many Eyeballs Does a Bug Need? An Empirical Validation of Linus' Law

Subhajit Datta[1], Proshanta Sarkar[2], Sutirtha Das[2], Sonu Sreshtha[2], Prasanth Lade[3], and Subhashis Majumder[2]

[1] Singapore University of Technology and Design
subhajit.datta@acm.org
[2] Heritage Institute of Technology
[3] Arizona State University

Abstract. Linus' Law reflects on a key characteristic of open source software development: developers' tendency to closely work together in the bug resolution process. In this paper we empirically examine Linus' Law using a data-set of 1,000+ Android bugs, owned by 70+ developers. Our results indicate that encouraging developers to work closely with one another has nuanced implications; while one form of contact may help reduce bug resolution time, another form can have quite the opposite effect. We present statistically significant evidence in support of our results and discuss their relevance at the individual and organizational levels.

Keywords: Linus' Law, Android, Connection, Betweenness, Social Network Analysis, Latent Dirichlet Allocation, Regression.

1 Introduction and Research Question

The *agile manifesto* announced in 2001, and the principles behind it emphasized on "individuals and interactions" in large scale software development[1]. Around the same time, Raymond's influential paper invoked the metaphor of the bazaar to highlight how myriad, spontaneous, and local interactions can fulfil global objectives in developing large and complex open source software [1]. In *Cathedral and the Bazaar* Raymond made a bold conjecture based on his observations of Linux development, calling it *Linus' Law*: "Given enough eyeballs, all bugs are shallow"; or more formally: "Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix will be obvious to someone" [1].

With the progressively empirical nature of software engineering research, anecdotal evidence needs to be complemented with statistically significant conclusions [2]. As is widely recognized, Linux is more than just an open source operating system; its significance lies in harnessing the agile methodology of software development in a truly novel way [1]. With this background, Linus'

[1] http://agilemanifesto.org/

G. Cantone and M. Marchesi (Eds.): XP 2014, LNBIP 179, pp. 242–250, 2014.

Law proclaims an interesting benefit of using agile methodologies by a wide and distributed developer pool. To validate whether Linus' Law captures merely a fortuitous quirk of Linux development, or has wider relevance in large scale agile development, we need to examine the law in a related but different development scenario. Android's[2] wide currency in today's computing milieu is indicated by the level of its usage in mobile computing devices and active developer pool [3]. Derived from the Linux kernel and having a similar development methodology, Android serves as an appropriate system for validating whether the claim of Linus' Law around the benefits of large scale agile practices indeed go beyond Linux. In this paper we present results from examining Linus' Law using Android bug report data.

In earlier examinations of Linus' Law using data from the Red Hat Enterprise Linux 4, the PHP programming language and the Wireshark network protocol analyzer, files with changes from nine or more developers were found to be 16 times more likely to have a vulnerability than files changed by fewer than nine developers [4], [5]. Linus' Law has also been called a "fallacy" due to the lack of supporting evidence [6]. These and similar other studies point to a lack of consensus on the validity as well as applicability of Linus' Law [7].

Our examination of Linus' Law using data from a large and widely used system has implications at several levels. For individual developers, our results can inform the benefits as well as costs of engaging closely with peers in the resolution of bugs. For managers, an understanding of Linus' Law and its limitations can be valuable for resource allocation. At the organizational level, our results can guide decisions on whether and how latest trends like crowdsourcing may help in bug resolution.

On the basis of the statements of Linus' Law mentioned in the previous section, we assume "eyeballs" to be a metaphor for focused developer attention on a bug, and a "shallow" bug is one which is resolved quickly. Thus Linus' Law is taken to propose that *bugs will be resolved faster if more developers attend to them.* The reference to "co-developer base" underscores a key expectation that developers engage in the bug resolution process *beyond* the immediate bugs they own. With this background, we arrive at our **research question**: *Does higher developer attention lead to Android bugs being resolved more quickly?*

For developers, we need to identify attributes that reflect on the level of their attention to resolving bugs. The spirit of agile development processes underlying Linus' Law encourages developers to engage across their peer group, sharing knowledge, expertise, and responsibilities [1]. We posit that for developers, the extent of connection and interpersonal influence in the project ecosystem is related to how quickly bugs are resolved. On the basis of these observations, we refine the research question into the following **hypotheses**:

- *H1: Developers who are more connected resolve their bugs more quickly.*
- *H2: Developers who have higher interpersonal influence resolve their bugs more quickly.*

[2] `www.android.com`

2 Methodology

Collecting Data: The Android bug reports data was accessed from a publicly available online repository [8]. The source XML file was parsed and the data persisted in a specifically designed MySQL database for easy querying. Each bug was identified by a unique bug identifier, and had the following attributes: title, status, owner, date opened, date closed, type, priority, component, stars, reported by, description. Each comment had the following attributes: identifier of the bug commented upon, commenter, date of comment, contents of comments.

Cleaning and Filtering Data: We calculated the *resolution time* for each bug as the number of days between date the bug was opened and the date it was closed. In the context of our study, we filtered the data by only considering bugs which have been commented by more than one developer. From this set we removed bugs with missing attributes or incorrectly recorded attributes (for example the opened date being later in time than the closed date). Finally we only considered bugs which had a resolution time of one year or less. We assume that a bug which has not been resolved for more than a year is unlikely to have attracted notable developer attention. Our final data-set consists of 1,016 bugs, and 73 unique developers who own at least one of these bugs. Each bug in this data-set has a unique owner; when we refer to a developer's bug(s) in subsequent discussion, we mean bug(s) which are owned by that developer.

Defining Developer Networks: We posit that developers can be connected at two levels as they work together to resolve these bugs: by co-commenting on bugs, which reflect shared interests and expertise, and through ownership of bugs which are related to one another. These two levels seek to capture the well recognized association between structure of work products and the structure of communication surrounding the work products [9]. To capture these two levels in our study, we construct the *developer communication network* (DCN) and *developer ownership network* (DON), whose vertices(nodes) are developers. In DCN two developers are connected by an edge (undirected link) if both have commented on at least one bug. For constructing DON we build an intermediate *bug similarity network* (BSN), whose vertices are bugs. In BSN, two bugs are connected by an edge if they are *similar* to one another by the measure explained below. In DON, two developers A and B are connected by an edge if there is at least one pair of bugs $bug_A - bug_B$ (bug_A owned by A and bug_B owned by B) such that bug_A and bug_B are joined by an edge in BSN.

In large software systems involving many developers such as Android, when a bug is raised its title and textual description are used to make a judgement on how similar it is to other bugs that have been addressed earlier [10]. On the basis of this judgement, the ownership of a bug gets decided; a bug is most likely to be assigned to a developer who has resolved similar bugs earlier. Thus a key step in the bug resolution process - assignment of ownership - is most often based on an evaluation of the similarity between bugs. Thus we can assume that developers owning bugs which are similar to one another are linked by a shared context.

Detecting Bug Similarity: To automatically detect similarities between bugs, we used a Latent Dirichlet Allocation (LDA) based approach. LDA considers a document to be a mixture of a limited number of topics and each word in the document can be attributed to one of these topics [11]. Given a corpus of documents, LDA discovers a set of topics, keywords associated with each of the topics and the specific mixture of these topics for each document in the corpus, and expresses these information as probability distributions [12]. In developing the LDA based topic models, we have used the collapsed Gibbs sampling method [12], [13].

Having obtained the probability distribution over topics for each bug, we calculate the similarity between all pairs of bugs in our data-set using the symmetric Kullback Leibler Divergence (KLD) [14]. KLD is a distance measure between two probability distributions. Since we seek to detect the most significant similarity between bugs (thereby reducing false positives to largest possible extent), we only connected two bugs by an edge in BSN if the corresponding KLD value was in the 96 to 100th percentile.

Examining Hypotheses: On the basis of the data-set and the two networks DCN and DON constructed as described above, we develop multiple linear regression models to examine the hypotheses, whose results are presented next.

3 Results and Discussion

We build regression models for the set of developers owning bugs to examine hypotheses H1 and H2. For the models we need to identify the *dependent variable*, the *independent variables*, and the *control variables*. The models will allow us to determine how the independent variables relate to the dependent variable, after accounting for the effects of the control variables.

3.1 Model Development

We now describe the development of the model for developers.

Independent Variables: To validate hypothesis H1, we need to identify a parameter that captures how much a developer is connected to his/her peers in the context of bug resolution. As defined in the Methodology section, DON captures how developers are linked to one another through the ownership of similar bugs. The *degree* of a developer in DON is the number of other developers (s)he is connected to via edges. As an established network metric, the degree of a vertex is a measure of the extent of its connection [15]. Thus we calculate the *Connection* of a developer as his/her degree in DON. For hypotheses H2, we need a measure of a developer's interpersonal influence in the collective enterprise of bug resolution. In social network analysis, the concept of *betweenness* reflects how important a person is as an intermediary in the flow of information between members of a network. Betweenness is measured by the *betweenness centrality* of a vertex, which is the proportion of all geodesics between pairs of

other vertices that include this vertex [15]. Individuals of higher betweenness are in stronger positions to broker the interaction of others. In our context, developers of high *Betweenness* in DON are expected to know more about a diverse range of bugs, and hence offer valuable guidance to other developers. On the basis of this background, *Connection* and *Betweenness* are considered as the independent variables in our model.

Dependent Variable: Both hypotheses H1 and H2 are concerned with how developers may resolve their bugs quickly. As a dependent variable in our model, we take the mean of the resolution time for all the bugs owned by a developer, denoted by *ResolutionTime*. The distributions of the resolution times of bugs owned by developers in our data-set have typically low skewness and kurtosis; thus the mean is a reasonably accurate measure of central tendency.

Control Variables: By developing the model, we expect to understand how *Connection* and *Betweenness* relates to *ResolutionTime*. However, to establish the relationship between independent variables and the dependent variable, we need to isolate some of the peripheral effects on the dependent variable. How much a developer can work on a bug to quickly resolve it, is influenced by how many bugs (s)he owns, or the total *Workload*. Additionally, since developers are encouraged to advice one another, a developer's *SpanOfInterest* - given by the number of bugs the developer has commented on - can also be expected to influence how quickly (s)he resolves his/her bugs. As defined in the Methodology section, DCN links developers through the co-commenting of work items. In social network analysis, clustering coefficient(CC)[3] measures how closely an individual is collaborating with others [15]. In our context, *CollaborationLevel* is the extent to which a developer is working with others and is a likely influence on how quickly (s)he resolves her bugs. Finally, we need to have a general sense of how much interest a bug has generated in the Android community. The "stars" field of bug report is "used in order to represent the number of people following a bug" [3]. The mean number of stars across all bugs owned by a developer - *CommunityConcern* - thus gives an indication of how much concern a developer's bugs have generated in the development ecosystem: a parameter that is likely to influence how quickly his/her bugs are resolved. With this background, we include *Workload*, *SpanOfInterest*, *CollaborationLevel*, and *CommunityConcern* as control variables in our model.

Model Assumptions and Variable Transformations: With reference to Table 1, column I gives the parameters of the *base model* which only considers the effects of the control variables, while column II reflects the *attention model* that additionally includes the independent variables. Multiple linear regression has the underlying assumptions of linearity, normality, and homoscedasticity of the residuals, and absence of multicollinearity between the independent variables.

[3] In a network, the clustering coefficient (C_v) for a vertex v is defined as follows: If v has a degree of k_v, that is there are k_v vertices directly linked to v, the *maximum* number of edges between these k_v vertices is k_v *choose* 2 or $k_v * (k_v - 1)/2$. If the *actual* number of such edges existing is N_v, then $C_v = 2 * N_v \ / \ k_v * (k_v - 1)$ [15].

The residual properties were verified using histogram, Q-Q plot and scatter plot of the standardized residuals. Among the variables, *Workload* and *SpanOfInterest* had a relatively high correlation (around 0.74), which is understandable as developers who own more bugs tend to comment more. Since the Variance Inflation Factors (VIF) of all variables were found to be below the upper limit of 10 in both the base and attention models [16], absence of appreciable multicollinearity was established. With references to the descriptive statistics in Table 1, although a skewness of around 3 for a variable is considered acceptable for including it in a linear regression model, we considered various established transformations for variables with relatively high values of absolute skewness, for making their distributions close to a normal distribution [16]. Accordingly, *Workload* and *CommunityConcern* variables were logarithmically transformed before including in the model. On the basis of the above discussion we concluded that the assumptions of linear multiple regression are valid within permissible limits in our case [16].

Model Description and Validation: In columns I and II of Table 1, the superscripts of the coefficients denote the range of their respective p values, as we specify in the table caption. The p value for each coefficient is calculated using the t-statistic and the Student's t-distribution. In the table's lower section, overview of the models are given: N denotes the number of data points used in building the model, in our case the number of developers who own bugs. R^2 is the coefficient of determination – the ratio of the regression sum of squares to the total sum of squares; it indicates the goodness-of-fit of the regression model in terms of the proportion of variability in the data-set that is accounted for by the model. *df* denotes the degrees of freedom. F is the Fisher F-statistic - the ratio of the variance in the data explained by the linear model divided by the variance unexplained by the model. The p value is calculated using the F-statistic and the F-distribution, and it indicates the overall statistical significance of the model. For the coefficients as well as the overall regression, if $p \leq$ *level of significance*, we conclude the corresponding result is statistically significant, based on null hypothesis significance testing.

From columns I and II of Table 1, we notice that by adding the independent variables, the R^2 value increases considerably between the base and attention models and the F-statistic also increases. Thus the independent variables have enhanced the explanatory power of the model. The standard technique of 10-fold cross validation was carried out by randomly partitioning the data into 10 sub-samples, training the interaction model with 9 sub-samples and validating the model on the 10th sub-sample, and repeating this procedure 10 times. The overall root mean square of prediction error from the cross validation process was found to be 74.53.

3.2 Threats to Validity

We report results from an observational study rather than a controlled experiment; thus in the statistical models developed, correlation does not imply

causation. Threats to **construct validity** arise from whether the variables are measured correctly. Although we have used established network metrics in our models, we recognize structures like DCN, BSN, and DON can be defined in other ways. We have used a LDA based approach for text similarity as simpler methods like cosine similarity do not consider clusters of keywords that are likely to occur together. We have also assumed that the elapsed time between bug opening and closure represents the actual time taken to resolve the bug. **Internal validity** ensures a study is free from systemic errors and biases. As the Android data-set is our only source of data, there is no notable threat to this type of validity. **External validity** is concerned with the generalizability of the results. We report results from studying only one data-set and the R^2 values of the models show there may be several other factors whose influence may not have been considered. We plan to address them in our future work. Thus we do not claim our results to be generalizable as yet. **Reliability** of a study is established when the results are reproducible. Given access to the Android bug report data, our results are reproducible.

Table 1. Left: Results of regression for the effects on bug resolution time.(Superscripts '***', '**', '†' denote $p \leq 0.0009$, $p \leq 0.001$, $p \leq 0.05$, respectively) **Right: Descriptive statistics for model variables.**

	I *Base model*	**II** *Attention model*	Mean	Stdev	Skew	Kurtosis
Dependent variable						
ResolutionTime			69.12	75.94	1.48	1.76
Intercept	81.60^{**}	144.03^{***}				
	(24.62)	(30.92)				
Control variables						
Workload	-27.30	27.42	13.92	26.17	3.57	14.87
	(20.24)	(42.35)				
SpanOfInterest	0.12	-0.004	53.44	12.19	2.33	4.18
	(0.01)	(0.11)				
CollaborationLevel	16.68	15.03	0.63	0.35	-0.5	-1.01
	(27.19)	(25.77)				
CommunityConcern	-17.30	-10.54	11.46	23.93	3.51	18.11
	(20.35)	(19.65)				
Independent variables						
Connection		-3.27^{**}	37.51	18.79	0.02	-1.12
		(1.11)				
Betweenness		$1.45^{\dagger}$	17.67	25.48	1.80	2.92
		(0.86)				
N	73	73				
R^2	0.05	0.17				
df	68	66				
F	0.89	2.3				
p	0.5	< 0.05				

3.3 Observations and Conclusions

On the basis of the details of the attention model in column II of Table 1, we can make a number of observations. The overall attention model as well as the relationship between both the independent variables and the dependent variable is statistically significant. From the sign of the respective coefficients, we notice that higher *Connection* for a developer relates to decreased *ResolutionTime* whereas higher *Betweenness* relates to increased *ResolutionTime*. Quicker resolution of bugs owned by a developer translates to lower *ResolutionTime*. Thus results from the model supports hypothesis H1, but we find evidence to contradict hypothesis H2.

Recalling that *Connection* is measured by the degree of a developer in DON, our results indicate that more connected a developer is to other developers through the ownership of similar bugs, (s)he is likely to be more deeply embedded in the development ecosystem, which is found to facilitate quicker resolution of the bugs owned by that developer. However, the more involved a developer gets in brokering interactions between other developers through his/her position of higher *Betweenness*, it appears that (s)he gets more distracted, which is reflected in the increased *ResolutionTime* of the bugs (s)he owns.

These results have notable implications in the development of large software systems. While developers need to be encouraged to connect directly with one another, the pitfalls of getting too engaged in facilitating interactions between other developers need to be recognized. As more complex software - many of them open source - is being built by larger teams, understanding the nuances of developer attention and its consequences is beneficial for individuals, management, and organizations. We can thus conclude that the broad assertion on more "eyeballs" making bugs "shallow" obscures important subtleties in the relationship between developer attention and how quickly bugs get resolved. While developers need to be encouraged to connect with one another as they collectively work on bug resolution, they also need to be sensitized to the challenges of too much involvement in mediating interactions between other developers.

References

1. Raymond, E.S.: The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly (2001)
2. Shaw, M.: Continuing prospects for an engineering discipline of software. IEEE Software 26, 64–67 (2009)
3. Guana, V., Rocha, F., Hindle, A., Stroulia, E.: Do the stars align? multidimensional analysis of android's layered architecture. In: 2012 9th IEEE Working Conference on Mining Software Repositories (MSR), pp. 124–127 (2012)
4. Meneely, A., Williams, L.: Secure open source collaboration: An empirical study of linus' law. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 453–462 (2009)
5. Meneely, A., Williams, L.: Strengthening the empirical analysis of the relationship between linus' law and software security. In: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2010, pp. 9:1–9:10. ACM, New York (2010)

6. Glass, R.L.: Facts and Fallacies of Software Engineering. Addison Wesley Professional, Pearson Education [distributor], Boston, Old Tappan (2002)
7. Wang, J., Carroll, J.M.: Behind linus's law: A preliminary analysis of open source software peer review practices in mozilla and python. In: 2011 International Conference on Collaboration Technologies and Systems (CTS), pp. 117–124. IEEE (May 2011)
8. Shihab, E., Kamei, Y., Bhattacharya, P.: Mining challenge 2012: The android platform. In: The 9th Working Conference on Mining Software Repositories (2012)
9. Conway, M.: How do committees invent?. Datamation Journal, 28–31 (April 1968)
10. Jeong, G., Kim, S., Zimmermann, T.: Improving bug triage with bug tossing graphs. In: ESEC/FSE 2009, pp. 111–120. ACM, New York (2009)
11. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. J. M. L. R. (March 2003)
12. Steyvers, M., Griffiths, T.: Probabilistic topic models. In: Latent Semantic Analysis: A Road to Meaning. Lawrence Erlbaum (2007)
13. Falessi, D., Cantone, G., Canfora, G.: Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques. IEEE Transactions on Software Engineering 39(1), 18–44 (2013)
14. Kullback, S., Leibler, R.A.: On information and sufficiency. Ann. Math. Statist. 22(1), 79–86 (1951)
15. Albert, R., Barabasi, A.: Statistical mechanics of complex networks. Cond-mat/0106096 (June 2001); Reviews of Modern Physics 74, 47 (2002)
16. Tabachnick, B., Fidell, L.: Using Multivariate Statistics. Pearson Education, Boston (2007)