An Empirical Study of Social Debt in Open-Source Projects: Social Drivers and the "Known Devil" Community Smell

H-M Chen, R. Kazman
University of Hawaii
{hmchen,kazman}@hawaii.edu

G. Catolino, M. Manca, D. Tamburri, W-J van de Heuvel
TU Eindhoven
{g.catolino, massimo.manca}@jads.nl, d.a.tamburri@tue.nl, W.J.A.M.v.d.Heuvel@jads.nl

Abstract

Social debt, the accumulation of unforeseen project costs from suboptimal human-centered software development processes, is an important dimension of technical debt that cannot be ignored. Recent research on social debt focusing on the detection of specific debt indicators, called community smells, has largely been conceptual and few of them are operationalizable. In addition, studies on the causes of community smells focused on group processes instead of individual tendencies. In this paper, we define and investigate four social drivers—factors that influence individual developer choices in their collaboration—in 13 open-source projects over four years: 1) inertia, 2) co-authorship (by chance or by choice), 3) experience heterophily, and 4) organization homophily. Building on previous studies and theories from sociology and psychology, we hypothesize how these drivers influence software quality outcomes. Our network analysis results include a contradiction to existing studies about experience heterophily and reveal a new community smell, which we call "Known Devil", that can be automatically detected.

Keywords: Technical Debt, Social Debt, Community Smells, Social Drivers, Social Network Analysis

1. Introduction

Research in technical debt has steadily received increased attention from academia and practice alike (Ernst et al., 2021; Google trend, 2023) since the term was created by Cunnigham in 1992 (Cunningham, 1992). It was initially a metaphor to explain software development actions and decisions akin to borrowing money to achieve goals. The research gradually progressed and expanded from software implementation concerns (at the code level) to the entire software engineering life cycle including requirements, design and architecture, implementation, testing, documentation, and deployment debt (Ernst et al., 2021; Li et al., 2015). Most importantly, the field has recently been further r expanded to include a project management aspect called social debt. Social debt has been studied in the social sciences for more than 50 years and is now borrowed by the software engineering community to describe an important and difficult challenge: the accumulation of unforeseen project costs from suboptimal human-centered software development processes (such as problems in coordination, collaboration, communication, and cooperation) and organizational structures (Caballero-Espinosa et al., 2023). Social debt has been shown to impact the quality outcomes of a software system in addition to impacting software teams, development processes and organizations (Tamburri et al., 2015; 2021).

The relatively recent inclusion of social debt into technical debt research is largely based on the socio-technical congruence (STC) hypothesis which builds on Conway's Law, that design structures and rules mirror the organizational structures of the organizations that produce them (Conway 1968; Conway 2023; Colfer & Baldwin, 2016; Sierra et al., 2018). Many researchers have studied how the alignment or misalignment of social communication structures and technical dependencies affect software quality. However, the results of the existing research were not conclusive and further research has been called for (Mauerer et al., 2022).

Other research has concentrated on identifying "community smells" which are sociotechnical anti-patterns, and sources of social debt (Caballero-Espinosa et al. 2023; Tamburri et al., 2021). So far there have been 30 community smells identified by the social debt research community (Caballero-Espinosa et al., 2023). Automation of the detection of these smells has been a focus as it allows more efficient social debt management. So far, only four of the 30 smells can be detected automatically in a software project (Tamburri et al., 2021). These are:

1) *organizational silo effect*: Siloed areas of the developer community that do not communicate, except through one or two of their respective

URI: https://hdl.handle.net/10125/107255
978-0-9981331-7-1

members, 2) *lone-wolf effect*: unsanctioned or defiant contributors who carry out their work with little consideration of their peers, their decisions and communications, 3) *black-cloud effect*: Information overload due to lack of structured communication or cooperation governance, and 4) *radio-silence effect*: leaders and teammates perform tasks in very formal and complex organizations. As such, team communication structures are not conducive to disseminating information across the teams efficiently.

Most importantly, the descriptions, which consist of the causes and effects of community smells, are primarily *conceptual***.** There are few empirical studies on the causes and effects of these smells, linking these to software quality measures. In particular, in the context of STC, what drives individual developers to behave, communicate or collaborate (e.g., we call social drivers) could affect the formation of community smells and social debt.

Our research aims to fill the void of existing research (see Figure 1: Research Framework). Based on our empirical investigation of 13 open-source projects, each spanning 4 years, our Phase 1 research (indicated in the yellow boxes in Figure 1 and presented in this paper) has identified four social drivers: 1) *Inertia*: in our study, the likelihood of selecting a past collaborator was almost 9 times higher than selecting a new potential candidate. 2) *co-authorship by design or by chance, working on the same file:* Developers were aware of those working on related tasks and tended to coordinate with them. 3***)*** *Experience heterophily*: different levels of expertise triggered conflicts in the processes for decision-making and for effectively sharing knowledge. 4) *Organization homophily* (developers being from the same organization) was, surprisingly, the least impactful social driver of the four.

In addition, our results from network analysis identified a new community smell: **"**Known Devil**"**. As we will show this smell is directly linked to a crucial software quality measure—bug rate—and can be automatically detected. These results and lessons learned from our phase 1 research provide a stepping stone for our Phase II research which focuses on STC and how community smells relate to technical dependencies and design smells and so provide insight to mitigate social debt.

In what follows, we present our research questions and theoretical foundation. In section 3, the research methodology for our Phase I research is described. In Sections 4-5, the data analysis, results and research limitation will be discussed. Section 6 concludes our Phase I research with comments on further research in our Phase II plan.

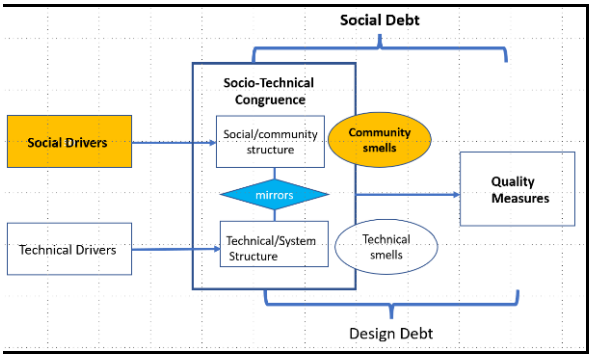

Figure 1. Research Framework

2. Related Study & Research Questions

Prior studies (Tamburri et al., 2015; 2019) have shown how social structure can have an impact on software project outcomes but fell short in terms of operationalizing the social relations and their impacts. In (Caballero-Espinosa et al. 2023), a total of 44 causes and 103 effects from the 30 community smells in the literature were extracted and categorized. The four most frequent types of causes include *context* (e.g., very formal processes, rigid procedures or standards), *communication* (poor or lack of), *composition* (of different expertise or professional background), and *coordination* (e.g., high task fragmentation, wasting time implementing modifications, increased isolation of teammates and disengaged developers).

The types of effects categorized in (Caballero-Espinosa et al. 2023) include *cooperation, technical debt effect, communication, coordination, cognition* (e.g., ignoring product requirements or architectural decisions), *economic effects, conflicts,* and *dual nature effects* which are technical problems that share traits of poor task coordination and conflicts. Cooperation was the most frequent type of effect, pointing out the importance of partnership, trust, and collective efficacy. The technical debt effect was the second most frequent type, which was identified in 13 out of 30 community smells. These results illustrate the close relationship between social debt and technical debt. Note that many already define social debt to be a type of technical debt (Ernst et al., 2021).

We aim to provide empirical evidence of some of the aforementioned causes and effects as well as finding the antecedents of the causes (e.g., social drivers) as shown in Figure 1. We focus on individual developer choices rather than group processes which were the focus of many prior studies. Two research questions are thus pursued:

RQ1: *What are the social drivers, e.g., motivation and strategies of individual developers that drive the exchange of information?*

RQ2: *How do the resulting communication strategies influence the software quality outcomes?*

In the search for social drivers which originate from the motivations and strategies of individual developers, an open-source context is appropriate for our study. In the open-source world, the social dynamics of software engineering are largely shaped by aggregating individual choices and those choices are influenced, in turn, by communication strategies (Conrad & Poole, 2012). Most of their communication strategy or behavior is carried out based on their need to improve their efficiency, as (Catolino et al., 2020) found. In addition, several psychological, organizational learning, and open system theories provide foundations for our hypotheses.

First, ***the law of inertia***, also called Newton's first law, states that: "A body at rest will remain at rest unless acted on by an unbalanced force" (OpenStax, 2023; Hanson,1963). This has been applied to group processes and termed "organizational inertia" (Godkin & Allcorn, 2008).. On the individual level, this refers to the tendency to perpetuate past behaviors. On one hand, inertia may generate high trust between software engineers, but on the other hand, it may prevent them from finding more efficient or effective collaborations. As a result, we can only postulate that inertia will affect performance outcomes.

H1*: the tendency to perpetuate past collaborations instead of creating new ones will impact performance.*

Second, homophily (McPherson, 2011) is the principle that interactions are more frequent between similar peers rather than dissimilar ones. When the communication happens mainly between developers belonging to the same organization, the benefits of the cooperation strategy may decrease. Therefore,

H2*: the tendency to collaborate within the organization is negatively associated with performance.*

Third, the open system theory states that all organizations are affected considerably by their environmental factors (Katz & Kahn, 1978). In the open-source world, choosing or being assigned to work on the same file by chance is a significant social driver in that developers are forced to communicate. If the collaboration is by choice, it may be due to inertia as hypothesized in H1. Otherwise, the forced communication by the coincidence of working on the same file may require more effort on the part of the collaborators, which may result in degraded performance. Hence,

H3*: Working with others on the same files is negatively associated with performance.*

Fourth, organizational learning, the process of creating and transferring knowledge within an organization, is influenced by its social structure. In particular, according to the participation framework view (Lave & Wenger, 1991), learning happens through firsthand experience enabled by social interactions. The study in (Škerlavaj et al., 2010) showed a prevailing communication between individuals similar in their level (homophily of experience) or belonging to a common organization (homophily) may create a barrier to information sharing. Thus, a heterophily strategy in experience and belonging could increase the project teams' knowledge and thus improve performance outcomes. Therefore,

H4*: the tendency to collaborate with others having different expertise levels is negatively associated with performance.*

3. Research Methodology

To operationalize the analysis of these hypotheses our first step was to retrieve and pre-process data from open-source version control systems and mailing lists. The selection of open-source projects for our investigation was motivated by the availability of appropriately rich data. The selection criteria include established reputation, size, having at least 4 years of history to mine and active mailing lists. We chose 13 large, well-known active open-source projects and for each of them extracted data covering a period of 4 years. The description of the projects are in Table 1, including average number of developers, commits and email per time window, number of organizations and absolute number of company developers in the whole period. Tenure is an approximation of the age in years of the project at the earliest date of analysis.

Table 1. Description of 13 Projects Selected

Name	Devs	Commits	Emails	Orgs.	Company devs	Tenure
Spark	208	422	3343	3	131	7
Ignite	82	432	2459	3	82	2.5
AmbarWe	46	456	2104	3	65	2
Cloudstack	98	1159	5343	3	95	2.5
Kafka	117	274	2308	3	112	6
Lenya	13	289	483	2	7	2
Cassandra	13	283	106	2	6	2
DPDK	144	610	4458	8	288	1.5
Couchdb	35	213	791	2	13	2.5
Drill	33	170	1253	2	39	2
Glusterfs	55	213	493	2	111	7
Qemu	204	954	10097	9	264	9
Libvir	39	330	1728	2	42	2

This dataset was cleaned and modeled as temporal edges as explained in sub-sections 3.1, 3.2, and 3.3. Different kinds of relationships—edges—were extracted from this dataset, such as collaboration and

communication. After the creation of the edge tables, they were plugged into the relational event model developed in the R package Goldfish. As explained in section 3.5, a fixed effect model was used to estimate the effect of engineers' behavior on performance, testing our hypothesis.

3.1. Data collection and pre-processing

The data sources are the log files extracted from the version control systems, as well as the mailing list archives and issue trackers of the 13 open-source projects. The mailing list and issue trackers contain the communication among software engineers (Bird et al., 2006). Prefixes from the email headers, such as 're', 'fwd', etc., were removed to improve matching email subjects. When we retrieved the revision history from each version control system, we ensured consistent naming by renaming files to use their most recent name. Similarly, as people may have multiple email addresses or nicknames, several heuristics were created to identify and connect them. After these heuristics were applied, the name groups and emails were manually inspected and corrected where necessary. The organizations involved in the project were determined by looking at the frequency of the email domains. When frequent domains of a company were found, they were researched to verify that the organization was indeed involved in the development.

Developer experience is calculated using a weighted sum of the commits where they appear either as authors or committers. The weight is the inverse of their age expressed in years. For example, a commit carried out 2 years earlier would have a weight equal to 0.5. The reason is that experience associated with old contributions tends to diminish as time goes by due to forgetting and due to project changes. A time window of 2 months was chosen as the basis for analysis. The authors and the committers within a given time window are classified as actors when they actively contributed to the mailing list or the source code. Committers, however, are counted even when they did not participate actively. This is because committers are usually involved longer in the project. However, if they disappeared for a long period—more than 2 time intervals—they were removed from the community. From the traces of commits and emails the communication and collaboration relationships were built. These are further discussed in the next sub-sections.

3.2. Temporal relationship modeling

Temporal networks are used when the connections between nodes arise and vanish dynamically. Hence, the edge table also includes the information of event time. In (Butts, 2008), the author introduced a framework to model the patterns in the sequence of events between actors over time; they called these temporal edges "relational events". This event consists of three elements: the sender, the receiver, and the time of occurrence. Relational event models are useful to capture the endogenous variables that influence the flow of interactions. For instance, actors in a network are usually bounded by social norms that influence their behavior. Using a network model allows us to test hypotheses about the agents' behavior in the network. A temporal view of the network is suitable because the open-source community has high turnover (Vasilescu et al., 2015) and the set of developers and the relationships between them change often over time. In this study, both collaboration and communication networks have temporal components described in more detail in the next sub-sections.

3.3. Temporal Collaboration Network

The heuristic we adopted to find collaboration is similar to those use by (Jermakovics et al. 2011), Lopez-Fernandez et al., 2004) and (Joblin et al., 2015). We determine collaboration between two authors if they have modified the same file. Measuring the correlation between the collaboration and the communication networks we found a statistically significant similarity using Quadratic Assignment Procedure (QAP) as illustrated by (Krackardt, 1987), (Anderson, 1999) and (Hubert & Arabie, 1989). QAP makes thousands of random permutations of the input graphs and measures the correlation between them. This creates a distribution of values that allows estimating the likelihood of the observed value in a series of random graphs. In most of the cases, it was found that the p-value was below 0.05, meaning that it was very unlikely to find a similar correlation between two random graphs.

As in (Joblin et al. 2015), the order of the commits was used to identify directed collaborations between authors: when one author co-changed a file after another a link is created from the former to the latter. However, after a long period, this relation is more tenuous. Hence a threshold of 45 days was set as the maximum amount of time between co-changes to consider it as a collaboration. Several thresholds were tried, such as 15, 20, 40, 55, 60 days, and 45 days was the threshold that produced collaboration networks with the highest correlation with the communication network. The list of relationships obtained with this method was used to build the collaboration network. The directions and the weights were removed because they were not important in this analysis.

3.4. Temporal Communication Network

Developers open and comment on threads to communicate with others. We considered a communication to be intentional when two or more developers commented on threads that have the same subject. An edge is created from the sender of an email to the previous senders of emails with the same subject. The communication edge list includes the sender, the receiver, and the date and time of the email. Emails are usually responded to in a couple of days; hence people who sent an email with the same subject are linked only if 7 days or fewer have passed. We experimented with different thresholds before choosing this number of days but the final graph did not change significantly.

3.5. Modeling with Dynamic Network Actor Models (DyNAMs) for Relational Events

DyNAMs (Stadtfeld, & Block, 2017) is a network model that estimates the probability of a relational event given the process state. The process state (yt) is the information at time t that describes the previous interactions. It includes the node-set, the previous relationships between nodes, the nodal attributes, and in some cases also exogenous variables, such as the presence of specific relationships between nodes. Previous relationships can be used to build statistics of the course of interactions. For example, statistics may take into account the number of emails sent from one person to another. This information may be important to predict a future interaction. Thus, the information collected at time t, yt, is used to predict the next relational event w. The coefficients of the statistics chosen are used to explain the behavior of the agents. For example, if developers sent a significantly higher number of emails to people belonging to the same organization, it is likely that the next event will be an email between developers of that organization. If the coefficients are statistically significant, they can predict the process of interactions and hypothesis testing about the behavior may be conducted with this approach. We can then test such hypotheses through the analysis of the significance of the coefficients as a linear regression.

DyNAMs is used in this study to explain the behavior in the exchange of emails between developers. It is reasonable to assume that the project members follow organizational strategies to improve their efficiency. Hence, the target variable is the communication link between any two pair of nodes in the network at a specific time. The DyNAMs' coefficients represent the developers' preferences in selecting a collaborator: their cooperation strategy. Once the developers' strategy is estimated using the DyNAMs model, a linear model is used to measure how their strategy (the DyNAMs' coefficients) affect the average number of bugs generated during development. The dataset uses observations at multiple time windows for all 13 projects.

3.6 Fixed-effect linear model

The DyNAMs coefficients represent the developers' preferences in selecting a collaborator. Thus, they represent the developers' cooperation strategy. Once the developers' strategy is estimated using the DyNAMs model, a linear model is used to measure how the developers' strategy (the DyNAMs coefficients) affect the average number of generated bugs during development. The dataset is a panel of data involving observation at several time windows of 13 projects. Thus, a fixed effect model is used to include individual differences among the projects. This implies including a different intercept for each project. The equation of the model (Equation 1) is:

$$log(BUGS) = w_{0i} + w_1 INERTIA_{it} + w_2 SAME\_ORGANIZATION_{it} + w_3 DIFF\_EXP_{it} + + w_4 CO\_AUTHORSHIP_{it} + w_5 TRANSITIVITY_{it} + + w_6 same\_organization\_se_{it} + w_7 \frac{commits_{it}}{c} + w_8 \frac{tenure_{it}}{c}$$

where the first four variables are the coefficients of the DyNAMs sub-model choice (Equation 2).

$$\beta^T s(i,j,y) = \beta_1 \times x_{inertia_i} + \beta_2 \times x_{same(organization)_i} + \beta_3 \times x_{diff(experience)_i} + + \beta_4 \times + x_{tie(collaboration)_i} + \beta_5 \times x_{trans_i} + \beta_6 \times x_{recip_i}$$

Transitivity (trans) pertains to the choice of a weak tie as a receiver, whereas reciprocity (recip) refers to the developers replying to others. The dependent variable is the logarithm of the number of commits that may have introduced bugs. The logarithm is used instead of the original values because the distribution was skewed with a long right tail. The logarithm transformation resulted in a bell-shaped distribution resembling a normal distribution.

To find the most suitable model several tests were performed. First, it was tested whether there were individual differences, the fixed effects, among projects. This was measured with an F-test which tests the hypothesis that all the individual intercepts of the projects are equal. Thus, the null hypothesis *H0* is: *w01 =w02 = ... = w13* while the alternative hypothesis *H1* is that *wxx* are not all the same.

Second, it was tested whether the error terms were correlated with regressors using the Hausman test. If that happens, the fixed effect estimates converge to the real values in large samples but not the random effects estimates. Therefore, if there is correlation between errors and regressors, the estimates of the models will differ significantly. The Hausman test's null hypothesis states that the fixed effect and random effect estimates are the same, which implies absence of correlation between errors and regressors. The alternative hypothesis states the opposite. If the null hypothesis is rejected, the fixed effect model is preferred while the random effect is preferred if the null hypothesis is accepted. In some of the projects the number of bugs was auto-correlated with its past values. Thus the Breusch-Godfrey test was used to test for auto-correlation. Breusch-Godfrey test's null hypothesis is the absence of serial correlation. A p-value less than the threshold of 0.05 brings to the rejection of the null hypothesis. Thus, the alternative hypothesis which states the presence of auto-correlation would be accepted. Several models with different variables and transformation were compared using the adjusted r squared.

4. Results and Analysis

4.1 DyNAMs interpretation

As mentioned, the DyNAMs' coefficients represent the developers' preferences in selecting a collaborator (e.g., social drivers). The results are summarized in Figure 2, ranking the four social drivers in order.

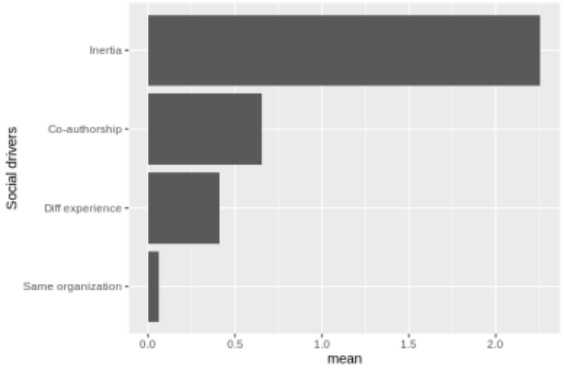

Figure 2: DyNAMs' coefficients mean values

1) ***Inertia (intertia_coeff)***: Most of the inertia effects were significantly positive and between the values of 1 and 4, with a mean of 2.19. This means that on average developers are almost 9 times more likely (exp(2.19)) to continue to communicate with past collaborators. This result showed that communication tended to flow frequently between the same persons.

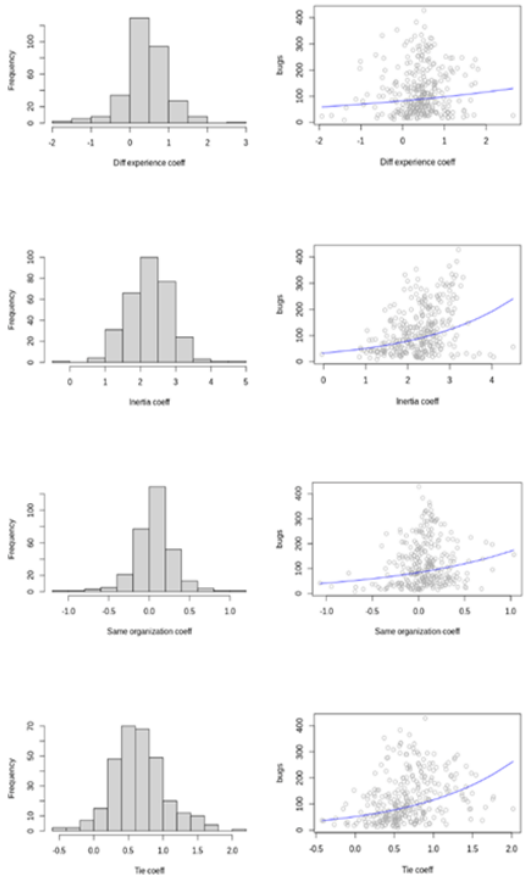

Figure 3: Coefficient distributions and their relationships to bugs

2) ***Co-authorship by design or by chance (tie_coeff):*** Working on the same file, on average, positively influenced communication choices. Developers may seek information from others working on the same file because they think their tasks should be coordinated with them. In such a case, they would need complementary information from them to accomplish their work. The average of the tie coefficients' values is 0.656. Therefore developers are 1.65 times more likely to choose to communicate with others working on the same file rather than someone working in another portion of the software.

3) ***Experience heterophily (diff_exp_coeff):*** Differences in experience influenced the choice of a collaborator. Developers with lower levels of experience sought information from those with greater levels. In open-source projects, it is quite common to have a small group of developers who produce the greatest amount of code. And in this study, it was found that these developers were more likely to

provide information to the less experienced ones, probably to mentor them. But the influence of the difference in expertise on choosing a cooperator is weak on average. Contrary to popular belief, we found that these developers were more likely to provide information to the less experienced ones, probably to mentor them. The "Diff experience" coefficient is on average 0.41, as shown in Figure 2.

4) ***Organization homophily (same_organization _coeff)***: Organization homophily coefficients are displayed in Figure 3 at an aggregate level. Here, it can be seen that most of the values are close to zero and a rather small portion are greater than 0.5 in absolute value. It can be deduced that developers, on average, do not consider belonging to the same organization relevant factor when deciding a collaborator.

4.2 Fixed-effect linear model results

As mentioned, a linear model is used to measure how the social drivers (the DyNAMs' coefficients) affect the average number of generated bugs during development and a fixed effect model is used to include individual differences among the projects. The coefficient w shown in the above equation of the model (Equation 1) represents the marginal effect of developers' behavior on the number of bugs they generated. Hence it measures how the number of bugs varies in contingency with the average behavior of developers. The model is log-linear and thus the slope and the elasticity changes at any point of y keeping the same sign of w; the coefficient can be interpreted as an approximation of the dependent variable's percentage variation when the predictor changes by 1 unit. The coefficient for each of the four social drivers were found as shown in Figures 4. Table 2 indicates that the four social drivers all influenced performance significantly.

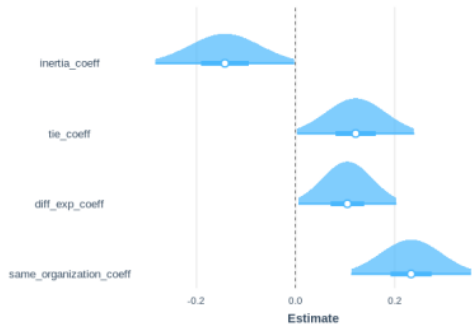

Figure 4: Fixed effect coefficients

Table 2.

	Dependent variable:	
	lbug	
	default	robust
	(1)	(2)
INERTIA	-0.143^{***}	-0.143^{**}
	(0.049)	(0.071)
CO-AUTHORSHIP	0.121^{**}	0.121^{***}
	(0.059)	(0.000)
DIFF_EXP	0.104^{***}	0.104^{***}
	(0.035)	(0.000)
SAME_ORG	0.233^{***}	0.233^{***}
	(0.071)	(0.000)
same_org_se	-2.019^{***}	-2.019^{***}
	(0.445)	(0.000)
TRANSITIVITY	-0.494^{**}	-0.494^{***}
	(0.196)	(0.000)
commits	0.186^{***}	0.186^{***}
	(0.009)	(0.000)
tenure	-0.009^{***}	-0.009^{***}
	(0.001)	(0.000)
Observations	308	308
R^2	0.728	0.728
Adjusted R^2	0.709	0.709
F Statistic (df = 8; 287)	96.218^{***}	96.218^{***}
Note:	*p<0.1; **p<0.05; ***p<0.01	

Model testing

p-value	method	alternative
0.001	F test	significant fixed effect
0.003	Hausman test	one model is inconsistent
<0.00	Breusch-Godfrey/Wooldridge	serial correlation in idiosync. errors

4.3 Hypotheses testing results

In this study we chose to use bug rate as the primary measure of software quality. The bug rates are determined by downloading each project's issue tracker data, selecting those labeled as "bug" and counting only those that had been resolved via a commit (and never reopened). The hypotheses testing results are summarized in Figure 3, with respect to bug rates.

H1 is supported: *the tendency to perpetuate past collaborations instead of creating new ones will impact performance*.

The coefficient can be interpreted as an approximation of the dependent variable's percent variation when the predictor changes by 1 unit. Therefore, a unitary increase in inertia corresponds to a 14.3% increase in bugs, on average. Repeated collaborations elicit social capital because the collaborators develop a common expertise that facilitates their understanding. In addition, it causes stability in the community and improves routines that regulate knowledge flows. It could happen that a strong inertia may prevent poor collaborations from dissolving, even when they should. Therefore, it seems quite likely that very high inertia values could hamper project performance. To measure this possibility, the square of the inertia coefficient was added to the

model. However, it was not significant and the model performance did not improve.

The Inertia coefficient was significantly different from zero. Hence, there is enough empirical evidence to reject the null hypothesis that inertia effect does not influence performance. The confidence intervals were large due to a high variance of the Inertia estimate. The coefficient's variance and therefore the estimate's accuracy will improve by collecting more data.

***H2* is supported:** *the tendency to collaborate within the organization is negatively associated with performance (increased bug rate).*

The fixed effect model shows that a unitary increase in organization homophily coefficient generates a 23.3% increase of bugs on average. However, most of the same organization's coefficients are around -0.5 and 0.5. Hence, it is more appropriate to talk about a 0.1 variation in the organizational homophily coefficient. A 0.1 homophily coefficient variation is associated with a 2.33% variation of the bugs in the same direction. The positive relationship between homophily coefficients and bugs suggests that the community will suffer when communication flows mainly between similar individuals in the same organization. This may result in a delay of information transfer from one organization to another.

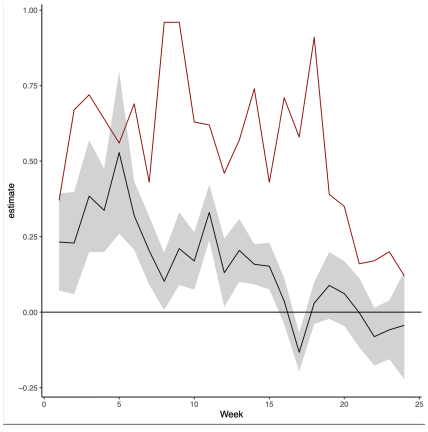

Figure 5: Organization homophily coefficients (in black) and buggy lines of code (in red) over time

H3 is supported*: Working with others on the same files is negatively associated with performance (increased bug rate).*

Developers tended to prefer cooperating and collaborating with others working on the same file. This behavior, however, had a significant impact on the bug rate. Tasks that are not based purely on a single file and which require coordination (for example, changing a file that other files depend on) are frequently not dealt with properly and hence generate more bugs. The scatter plot in the fourth row of Figure 3 shows a positive relationship between the coefficient values and the number of bugs. This suggests that the preference to communicate with co-authors of the same file may create tunnel vision. This may lead to inadequate communication to modify and to integrate files without causing bugs. The co-authorship coefficient w2 is 0.121. Adding the square to test for a decreasing marginal effect did not improve the model.

When collaborations tend to get stronger among those co-authoring the same files and weaker with others working in different files, errors are more frequent. This suggests that tasks that are not included on the same file but needing coordination are not dealt with properly and generate bugs.

H4 is not supported: *the tendency to collaborate with others having different expertise levels is positively associated with performance (decreased bug rate).*

Contrary to expectations, the number of bugs developers generate is positively associated with their tendency to communicate at higher rates with those dissimilar in experience. The theory of situated learning through social contacts with dissimilar individual does not find confirmation in our study. A possible alternative explanation could be that developers tend to acquire more useful and understandable information from those that have a similar level of expertise.

5. Discussion

First, from our results, a community smell we call "Known Devil" is identified. The tendency to perpetuate past collaborations instead of creating new ones due to inertia is an intent to minimize risk, as the saying goes: *"Better the devil you know than the devil you don't."* However, the choice to avoid the risk of working with a new collaborator may result in higher bug rates especially when they work on the same files—our study defines collaboration as working on the same file. Inertia prevents finding the optimal collaborators and may create inefficiencies and errors as hypothesized in H3. This community smell is therefore associated with two social drivers and can be automatically detected as devised in our study.

Second, H4 not being supported seems to be strongly related to the "Known Devil" community smell we identified. This is because we assume that prior collaborators had the same or similar backgrounds and experience levels. However, this assumption may not hold true. This calls for further investigation.

Third, our results indicate working on the same file is a strong social driver. However, further analysis is needed to separate co-authorship *by chance* from the co-authorship *by choice* in working on the same file; it is difficult to distinguish the two in the open-source environment. Does the collaboration start first and then the collaborators decide to choose to work on the same file? Or the other way around?

Fourth, the findings of H3—that working on the same file is associated with higher bug rate—needs to be further studied in our Phase II research, as indicated in Figure 1. We hypothesize that there are technical dependencies that are beyond simply collaborating on files. This suggests that there exists a misalignment of social structure vs. technical structure which causes the increased bug rate. This will be investigated in our Phase II research in which we will also expand the outcome measurement beyond bug rate, including team productivity, developer satisfaction, developer intention to stay, and other software quality measures.

Finally, our current Phase I research has the following limitations:

a. This analysis focused on 308 observations of 13 open-source projects. Thus, the generalization of the results may be limited.
b. Data about communication were retrieved from mailing lists. However, developers have other communication channels, e.g. personal contacts, private emails, etc. This communication could not be observed. However, it is reasonable to assume that most of the communication happened within the official channel, that is the mailing list, in an open-source environment.
c. As mentioned, the values of organizational homophily were limited to a small range. Therefore, these findings on homophily effects cannot be generalized with high accuracy for cases of high homophilous behaviors.
d. The communication and collaboration network construction could have an impact on results. However, the method used here was based on past research and additional statistical analyses.

Useful enhancements to this research program include: 1) distinguishing between different activities, such as bug fixing, documentation or code contributions, etc. to understand the decision-making process and division of tasks, and 2) classifying the content of the emails using text mining to understand the kinds of information exchanged.

6. Conclusions

Previous studies on social debt, and social smells in particular, are conceptual in nature and focus on group processes instead of individual tendencies. Our study helps to discover the antecedents (causes) of social debt on the individual developer level—social drivers—and their effects on software quality (bugs). Built on theories in sociology, systems, organizations, and psychology, we identified four social drivers (inertia, co-authorship by design or by chance, experience heterophily, and organization homophily) which were all found to significantly influence developers' choices in finding a collaborator. For our investigation, we chose 13 large, well-known open-source projects and for each of them extracted data covering a period of 4 years. Employing the Dynamic Network Actor Models (DyNAMs) and a fixed-effect linear model, we tested four hypotheses. We found:

- H1 is supported: the tendency to perpetuate past collaborations instead of creating new ones will impact performance. This study shows that the impact is negative.
- H2 is supported: the tendency to collaborate within the organization is negatively associated with performance.
- H3 is supported: Working with others on the same files is negatively associated with performance.

Contrary to previous studies and existing theories:

- H4 is not supported: the tendency to collaborate with others having different expertise levels is positively associated with performance.

From these results, the "Known Devil" community smell has emerged. As the saying goes "*Better the devil you know than the devil you don't.*" The tendency to perpetuate past collaborations instead of creating new ones due to inertia may result in higher bug rates especially when they work on the same files. This community smell is associated with two social drivers and can be automatically detected as devised in our study. Many mitigation strategies (Ernst et al., 2021, pp. 208-210) can, however, be devised for avoiding the Known Devil social smell which may increase social debt. A few actions are recommended: 1) Raise awareness of this smell and one's tendency to work with "known devils". 2) Monitor collaborations on files so that individual developers can be made known of their "known devils". This is the benefit of being able to detect smells automatically. 3) Create a broader list of potential collaborators for developers to choose from. 4) Design social-networking opportunities for developers to get to know more people who could be potential collaborators.

The present study has a few limitations and cannot be generalized outside the open-source project context. Nevertheless, as depicted in our research framework in Figure 1, it contributes to illuminate an important research direction in understanding social

debt, which is an integral part of technical debt that needs to be well-managed in any modern organization.

7. Acknowledgments

This project was partially supported by the European Union's Horizon Europe research and innovation program (grant #101073945; SAFE-CITIES) and the National Science Foundation award #2232721.

8. References

Anderson, B. S., Butts, C., & Carley, K. (1999). The interaction of size and density with graph-level indices. Social networks, 21(3), 239-267.

Bird, C., Gourley, A., Devanbu, P., Gertz, M., & Swaminathan, A. (2006). Mining email social networks. In Proc. 2006 Intl workshop on Mining software repositories, 137-143.

Butts, C. T. (2008). A relational event framework for social action. Sociological Methodology, 38(1), 155-200.

Caballero-Espinosa, E., Carver, J. & Stowers, K. (2023). Community smells—The sources of social debt: A systematic literature review. Information and Software Technology, 153.

Catolino G., Palomba F., Tamburri, D. A., Serebrenik, A. Ferrucci. F. (2020). Refactoring community smells in the wild: the practitioner's field manual. Proc. 42nd Intl Conference on Software Engineering, 25–34.

Colfer, L. J., & Baldwin, C. Y. (2016). The mirroring hypothesis: theory, evidence, and exceptions. Industrial and Corporate Change, 25(5), 709-738.

Conrad C. & Poole, M.S. (2012). *Strategic organizational communication: In a global economy.* Wiley &Sons.

Conway, M. E. (1968). How do Committees Invent?. Datamation, 14 (5), 28–31.

Conway, M. E. (2023) Conway's Law. https://www.melconway.com/Home/Conways_Law.html)

Cunningham, W. (1992) The WyCash Portfolio Management System. OOPSLA '92 Experience Report. http://c2.com/doc/oopsla92.html

Ernst, N., Delange, J., & Kazman, R. (2021). *Technical Debt in Practice—How to Find It and Fix It*. MIT Press.

Godkin, L., & Allcorn, S. (2008). Overcoming organizational inertia: A tripartite model for achieving strategic organizational change. J. Applied Business and Economics, 8(1), 82.

Google trend. (2023). Technical debt trend 2004-2023. https://trends.google.com/trends/explore?date=all&geo=US&q=technical%20debt&hl=en

Hanson, N, R. (1963) The Law of Inertia: A Philosopher's Touchstone. Philosophy of Science, 30 (2).

Hubert, L, & Arabie, P. (1989). Combinatorial data analysis: Confirmatory comparisons between sets of matrices. Applied stochastic models and data analysis, 5(3), 273-325.

Jermakovics, A., Sillitti, A., & Succi, G. (2011). Mining and visualizing developer networks from version control systems. In Proc. of 4th Intl Workshop on Cooperative and Human Aspects of Software Engineering, 24-31.

Joblin, M., Mauerer, W., Apel, S., Siegmund, J., & Riehle, D. (2015). From developer networks to verified communities: A fine-grained approach. In 2015 IEEE Intl Conference on Software Engineering, 563-573).

Katz, D., & Kahn, R. L. (1978). Organizations and the system concept. Classics of organization theory, 80.

Krackardt, D. (1987). QAP partialling as a test of spuriousness. Social networks, 9(2), 171-186.

Lave, J., & Wenger, E. (1991). Situated learning: Legitimate peripheral participation. Cambridge University Press.

Li, Z., Avgeriou, P., & Liang, P. (2015). A systematic mapping study on technical debt and its management. J. Systems and Software, 101, 193-220.

Lopez-Fernandez, L., Robles, G., & Gonzalez-Barahona, J. M. (2004). Applying Social Network Analysis to the Information in CVS Repositories. Proc. MSR 2004.

Mauerer, W., Joblin M., Tamburri, D., Paradis, C., Kazman R., & Apel, S. (2022). In Search of Socio-Technical Congruence: A Large-Scale Longitudinal Study, IEEE Trans. Software Engineering, 48(8), 3159-3184.

McPherson, M., Smith-Lovin, L., & Cook J.M. (2001). Birds of a feather. Homophily in social networks. Annual review of sociology, 27(1), 415–444.

Openstax. Newton's first law. https://openstax.org/books/university-physics-volume-1/pages/5-2-newtons-first-law.

Paradis, C. & Kazman, R. (2021). Building the MSR tool Kaiaulu: Design Principles and Experiences, Proc. European Conference on Software Architecture (ECSA) 2021, 107-129.

Sierra, J. M., Vizcaíno A, Genero, M. & Piattini, M (2018). A systematic mapping study about socio-technical congruence, Information and Software Technology (94), 111-129.

Škerlavaj, M., Dimovski, V., & Desouza, K. C. (2010). Patterns and structures of intra-organizational learning networks within a knowledge-intensive organization. J. Information Technology, 25(2), 189-204.

Stadtfeld, C., & Block, P. (2017). Interactions, actors, and time: Dynamic network actor models for relational events. Sociological Science, 4, 318-352.

Tamburri, F., Kruchten P., Lago P & van Vliet H. (2015). Social debt in software engineering: insights from industry. J. Internet Services and Applications, 6(10).

Tamburri, D., Kazman, R. & Fahimi, H. (2016). The Architect's Role in Community Shepherding. IEEE Software, 33 (6), 70-79.

Tamburri D., Palomba, F. & Kazman, R. (2021). Exploring Community Smells in Open-Source: An Automated Approach. IEEE Trans. Software Engineering, 47(3), 630-652.

Tamburri, D., Kazman, R., & Fahimi, H. (2022). On the Relationship Between Organizational Structure Patterns and Architecture in Agile Teams", IEEE Trans. Software Engineering, 49 (1), 325-347.

Vasilescu, B., Filkov, V., & Serebrenik, A. (2015). Perceptions of diversity on git hub: A user survey. 8th Intl Workshop on Cooperative and Human Aspects of Software Engineering, 50-56.