What Improves Developer Productivity at Google? Code Quality

Lan Cheng
Google
United States
lancheng@google.com

Emerson Murphy-Hill
Google
United States
emersonm@google.com

Mark Canning
Google
United States
argusdusty@google.com

Ciera Jaspan
Google
United States
ciera@google.com

Collin Green
Google
United States
colling@google.com

Andrea Knight
Google
United States
aknight@google.com

Nan Zhang
Google
United States
nanzh@google.com

Elizabeth Kammer
Google
United States
eakammer@google.com

ABSTRACT

Understanding what affects software developer productivity can help organizations choose wise investments in their technical and social environment. But the research literature either focuses on what correlates with developer productivity in ecologically valid settings or focuses on what causes developer productivity in highly constrained settings. In this paper, we bridge the gap by studying software developers at Google through two analyses. In the first analysis, we use panel data with 39 productivity factors, finding that code quality, technical debt, infrastructure tools and support, team communication, goals and priorities, and organizational change and process are all causally linked to self-reported developer productivity. In the second analysis, we use a lagged panel analysis to strengthen our causal claims. We find that increases in perceived code quality tend to be followed by increased perceived developer productivity, but not vice versa, providing the strongest evidence to date that code quality affects individual developer productivity.

CCS CONCEPTS

• **Software and its engineering** → **Software creation and management**; *Software development process management*; • **General and reference** → **Empirical studies**.

KEYWORDS

Developer productivity, code quality, causation, panel data

ACM Reference Format:
Lan Cheng, Emerson Murphy-Hill, Mark Canning, Ciera Jaspan, Collin Green, Andrea Knight, Nan Zhang, and Elizabeth Kammer. 2022. What Improves Developer Productivity at Google? Code Quality. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3540250.3558940

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9413-0/22/11.
https://doi.org/10.1145/3540250.3558940

1 INTRODUCTION

Organizations want to maximize software engineering productivity so that they can make the best software in the shortest amount of time. While software engineering productivity "is essential for numerous enterprises and organizations in most domains" [50] and can be examined through multiple lenses [29], understanding the productivity of individual software developers can be especially fruitful because it has the potential to be improved through many actions (e.g. from tooling to process changes) and by many stakeholders (from individual developers to executives). However, it is difficult to know which actions will truly improve productivity in an ecologically valid setting, that is, in a way that accurately characterizes productivity in a realistic software development context. This motivates our research question:

> **RQ:** What causes improvements to developer productivity in practice?

A wide spectrum of prior research provides some answers to this question, but with significant caveats. For example, at one end of the research spectrum, Ko and Myers' controlled experiment showed that a novel debugging tool called Whyline helped Java developers fix bugs twice as fast as those using traditional debugging techniques [30]. While this evidence is compelling, organizational leaders are faced with many open questions about applying these findings in practice, such as whether the debugging tasks performed in that study are representative of the debugging tasks performed by developers in their organizations. At the other end of the spectrum, Murphy-Hill and colleagues surveyed developers across three companies, finding that job enthusiasm consistently correlated with high self-rated productivity [36]. But yet again, an organizational leader would have open questions about how to apply these results,

Table 1: Hypothetical cross sectional survey response data.

	Productivity Rating	Code Quality Rating
Aruj	Somewhat productive	Medium quality
Rusla	Extremely productive	Extremely high quality

Table 2: More hypothetical survey responses, collected 3 months after the data in Table 1. Plusses (+) and minuses (–) indicate the direction of the change since the prior survey.

	Productivity Rating	Code Quality Rating
Aruj	Highly productive (+)	High Quality (+)
Rusla	Somewhat productive (–)	High Quality (–)

such as whether some unmeasured third variable causes *both* high productivity and high job enthusiasm.

More broadly, these examples illustrate the fundamental limitations of prior approaches to understanding developer productivity. On one hand, software engineering research that uses controlled experiments can help show with a high degree of certainty that some practices and tools increase productivity, yet such experiments are by definition highly controlled, leaving organizations to wonder whether the results obtained in the controlled environment will also apply in their more realistic, messy environment. On the other hand, research that uses field studies – often with cross sectional data from surveys or telemetry – can produce contextually valid observations about productivity, but drawing causal inferences from field studies that rival those drawn from experiments is challenging.

Our study builds on the existing literature about developer productivity, contributing the first study to draw strong causal conclusions in an ecologically valid context about what affects individual developer productivity.

2 MOTIVATION

The paper's main technical contribution – the ability to draw stronger causal inferences about productivity drivers than in prior work – is enabled by the use of the *panel data analysis* technique [25]. In this section, we motivate the technique with a running example.

Much of the prior work on developer productivity (Section 3) relies on *cross-sectional data*. To illustrate the limitations of cross sectional data, let us introduce an example. Consider a survey that asks about respondents' productivity and the quality of their codebase. The survey is distributed at a large company, and two developers respond, Aruj and Rusla. Let's assume their responses are representative of the developer population. Their survey responses are shown in Table 1.

From this survey, we see that productivity correlates with code quality. But we cannot confidently say that high code quality causes high developer productivity, due in part to the following confounding explanations [1]:

- **Time-invariant effects.** These are effects that have the same influence over time. For example, if Rusla went to college and Aruj did not, from cross-sectional data, we cannot distinguish between the effect of college and the effect of code quality on productivity.
- **Respondent-independent time effects.** These are effects that influence all respondents uniformly, such as seasonal effects or company-wide initiatives. For example, prior to the survey, perhaps all engineers were given their annual bonus, artificially raising everyone's productivity.
- **Non-differentiated response effects.** These are effects where respondents will give the same or similar responses to every survey question, sometimes known as straightlining. For example, perhaps Aruj tends to choose the middle option to every question and Rusla tends to answer the highest option for every question.

We use panel analysis to address these confounds, enabling stronger causal inference than what can be obtained from cross sectional data [25]. The power of panel data is that it uses data collected at multiple points in time from the same individuals, examining how measurements change over time.

To illustrate how panel data enables stronger causal inference, let us return to the running example. Suppose we run the survey again, three months later, and obtain the data shown in Table 2.

One interesting observation is that if we analyze Table 2 in isolation, we notice that there's not a correlation between productivity and code quality – both respondents report the same code quality, regardless of their productivity. But more importantly, looking at the *changes* in responses from Table 1 and Table 2, we see productivity changes are now correlated: Aruj's increasing productivity correlates with increasing code quality, and Rusla's decreasing productivity correlates with decreasing code quality.

Panel analysis rules out the three confounding explanations present in the cross-sectional analysis:

- In the cross-sectional analysis, we could not determine if Rusla's high productivity was driven by her college education. But in this panel analysis, we can rule out that explanation, because college is a time invariant exposure – it theoretically would have the same effect on her productivity in the first survey as in the second survey. This ability to rule out other potential causes that are time invariant exists whether or not the researcher can observe those potential causes. While with cross-sectional analysis, researchers may be able to control for these potential causes using control variables, researchers have to anticipate and measure those control variables during analysis. This is unnecessary with panel analysis because time invariant factors are eliminated by design.
- In the cross-sectional analysis, we could not determine if productivity was driven by a recent annual bonus. This explanation is ruled out in the panel analysis. If the annual bonus had an effect, the change in productivity scores across both participants would be uniform.
- In the cross-sectional analysis, we could not determine if respondents were just choosing similar answers to every question. This explanation is also ruled out with panel analysis. If respondents were choosing similar answers, there would be no change in productivity scores, and thus we would not see a correlation among the changes.

The ability of panel analyses to draw relatively strong causal inferences makes it a quasi-experimental method, combining some of the advantages of experiments with those of field studies [24].

3 RELATED WORK

To answer research questions similar to ours, several researchers previously investigated what factors correlate with developer productivity. Petersen's systematic mapping literature review describes seven studies that quantify factors that predict software developer productivity [39], factors largely drawn from the COCOMO II software cost driver model [10]. For instance, in a study of 99 projects from 37 companies, Maxwell and colleagues found that certain tools and programming practices correlated with project productivity, as measured by the number of lines of written code per month [33]. More broadly, a recent study explored what factors correlate with individual developers' self-reported productivity at three companies [36]. In contrast to our study, these prior studies report correlations with relatively weak causal claims.

Other researchers have been able to make stronger causal claims about programmer productivity by running controlled experiments. For instance, when Tosun and colleagues asked 24 professionals to complete a simple programming task, either using test-driven development (treatment group) or iterative test-last development (control group), they found that treatment group participants were significantly more productive than control group participants, where productivity was measured as the number of tests passed in a fixed amount of time [49]. Such randomized controlled experiments are considered a "gold standard" because they can make very strong causal claims [8]. The challenge with such studies is that they are expensive to run with high ecological validity. Consequently, such studies typically use students as participants rather than professionals (e.g. [13, 44, 46]), use small problems and programs rather than more realistic ones (e.g. [5, 35, 46]), and can only vary one or two productivity factors per experiment (e.g. [14, 31, 42]). While the study presented here cannot make as strong causal claims as experiments, the present field study has higher ecological validity than experimental studies.

To address these challenges, software engineering researchers have been using causal inference techniques in field studies, where stronger causal claims can be made than in studies with simple correlations. The core of such studies is analyses that leverage time series data, rather than cross-sectional data. For instance, Wang and colleagues use Granger's causality test [20] to infer that women's pull requests cause increases in those women's follower counts [51]. As another example, using the Bayesian CausalImpact framework [6], Martin and colleagues show that 33% of app releases caused statistically significant changes to app user ratings [32]. These papers used fine-grained time series data, which is not possible for the type of data described in this paper, and to our knowledge, has not been applied to studies of developer productivity.

Panel analysis, another causal inference technique, has been used by prior software engineering researchers. Qiu and colleagues used GitHub panel data to show that "social capital impacts the prolonged engagement of contributors to opensource" [40]. Islam and colleagues used panel data to show that distributed version control systems "reduce the private costs for participants in an OSS project and thus increases the number of participants, but decreases the average level of contribution by individual participants" [26]. Like these papers, we use panel data to make causal inferences, but in our case, the inferences are about developer productivity.

4 PANEL ANALYSIS: METHODS

Towards answering our research question, we next describe our data sources, dependent variables, independent variables, panel data, and modeling design.

4.1 Data Sources

The data of this study comes from two sources: Google engineers' logs data and a company-wide survey at Google. Neither source was built specifically for the research we describe here, and so we consider this opportunistic research.

4.1.1 Logs Data. We collected a rich data set from engineers' logs from internal tools, such as a distributed file system that records developers' edits, a build system, and a code review tool. This data contains fine-grained histories of developers' work, enabling us to make accurate measurements of actual working behavior, such as the time developers spend actively writing code. The data helps us characterize developers' work practices, such as what kinds of development tasks they are doing, how long those tasks take, and how long they are waiting for builds and tests to complete. Details on these tools, how data is aggregated into metrics, measurement validation, and ethical considerations of data collection can be found elsewhere [27]. We describe the exact metrics we use in Section 4.3.

4.1.2 EngSat. The Engineering Satisfaction (EngSat) Survey is a longitudinal program to: understand the needs of Google engineers; evaluate the effectiveness of tools, process, and organization improvements; and provide feedback to teams that serve Google engineers. Every three months, the survey is sent out to one-third of eligible engineers – in one of five core engineering job roles, have been at Google's parent company for at least 6 months, and below the "director" level. The same engineers are re-surveyed every three quarters, and a random sample of one-third of new engineers is added each quarter so that all engineers are invited. The survey questions cover a range of topics, from productivity to tool satisfaction to team communication. Respondents are asked to describe their experience in the 3 month period prior to taking the survey. Before beginning the survey, respondents are informed how the data is used and that participation is voluntary.

While EngSat response rates are typically between 30% and 40%, response bias does not appear to be a significant threat. We know this because we analyzed two questions for response bias, one on productivity and one on overall engineering experience satisfaction. We found that EngSat tends to have lower response rates for technical leads and managers, those who have been at Google for a longer period, and for engineers from the San Francisco Bay Area, where Google is headquartered. To estimate the impact of non-response bias on a metric derived from EngSat responses, we compare a bias-corrected version of the metric to its uncorrected version and check for the difference. The bias-corrected metric is calculated by reweighting EngSat responses with the propensity score (a similarity measure [21]) of responding to EngSat, which is

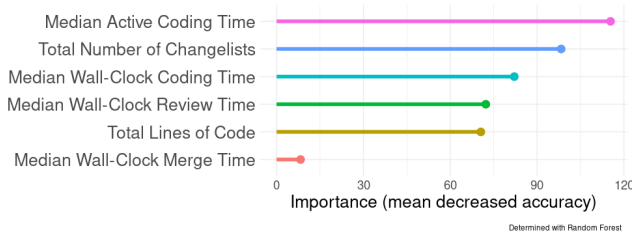

Figure 1: Quantitative features that predicted self-rated productivity.

estimated based on factors such as developer tenure, work location, and coding language and tools. We find that after correcting for this non-response bias using propensity score matching, the percent of engineers who responded favorably did not change significantly for either question. For instance, adjusting for non-response bias, productivity decreases relatively by 0.7%, which is too small to reach statistical significance at the 95% level. These results were consistent across the three rounds of EngSat that we analyzed.

4.2 Dependent Variable: Productivity

We use self-rated productivity from our survey as our dependent variable. While subjective and objective measures of productivity each have advantages and disadvantages, we chose a subjective measure of productivity here both because it is straightforward to measure in survey form and because it is used broadly in prior research [15, 34, 36].

The EngSat survey asks the question: *Overall, in the past three months how productive have you felt at work at Google/Alphabet?* Respondents can choose "Not at all productive", "Slightly productive", "Moderately productive", "Very productive", or "Extremely productive". We coded this variable from 1 (Not at all productive) to 5 (Extremely productive).

Prior software engineering research has shown that subjective productivity correlates with objective measures of productivity [36, 37] as a way to establish convergent validity of question-based productivity metrics (that is, how they relate to other measures of the same construct [7]), We sought to do the same by correlating our subjective productivity measure below with several objective measures of productivity. Rather than using a linear correlation as used in prior work, we were open to the possibility that relationships were non-linear, and thus we selected a random forest as a classifier.

First, we created a simple random forest to predict a binary version of self-rated productivity, where we coded "Extremely productive" and "Very productive" as productive, and the other values as not productive. We then predicted this binary measure of self-rated productivity using six quantitative productivity metrics measured over a three month period. Two of the measures capture the amount output produced over the fixed period:

- *Total Number of Changelists.* This represents the number of changelists (CLs) that an engineer merged, after code review, into Google's main code repository.
- *Total Lines of Code.* Across all CLs an engineer merged, the total number of lines of code changed.

Two measures capture the amount of time it takes an engineer to produce one unit of output (a changelist):

- *Median Active Coding Time.* Across every CL merged, the median time an engineer spent actively writing code per CL [27].
- *Median Wall-Clock Coding Time.* The median wall-clock time an engineer spent writing code per CL, that is, the time elapsed between when the engineer starts writing code and when they request the code be reviewed.

The remaining two measures captured non-productive activities, that is, how much time an engineer spends waiting per unit of output (a changelist):

- *Median Wall-Clock Review Time.* The median wall-clock time an engineer spent waiting for code review per CL.
- *Median Wall-Clock Merge Time.* The median wall-clock time an engineer waited between approval for merging and actually merging per CL.

We gathered the above data over 6 consecutive quarters from 2018Q1 to 2019Q2. For each quarter, we linked an engineer's subjective measure of productivity to the above five quantitative measures. Since engineers are invited to take our survey once every 3 quarters, a single engineer may be represented at most twice in this data set. In total, we had 1958 engineer data points for our model.

After randomly selecting 10% of the data for validation, the model had 83% precision and 99% recall, suggesting a substantial relationship between quantitative and qualitative productivity measures. Looking at the importance of each quantitative metric in classifying developers in the model (Figure 1), we see that Median Active Coding Time was the most predictive quantitative feature. This aligns with Meyer and colleagues' finding that Microsoft engineers view coding as their most productive activity [34].

4.3 Independent Variables

To predict the dependent variable, we started with 42 independent variables – reduced to 39 after a multicolinearity check (Section 4.6) – available from the survey and logs data. Since survey respondents are asked to report on their experiences from the three months prior to the survey, we collected log data for the corresponding three month period. While many metrics could be analyzed, we selected metrics that were relatively straightforward to collect and that appeared plausibly related to individual productivity, based on consultation with internal subject matter experts within Google that were experienced with building and deploying developer metrics.

Below, we group independent variables into six categories, describe each variable, and link them to prior work. We give each variable a short name (in parentheses) to make referencing them easier in the remainder of the paper. Full survey questions and response scales are available in the Appendix.

4.3.1 Code Quality & Technical Debt. The first category of potential drivers of productivity are those relating to code quality and technical debt. Based on experience, DeMarco and Lister claim that software quality, generally speaking, "is a means to higher productivity" [11]. In an experiment, Schankin and colleagues found that participants found errors 14% faster when descriptive identifier names were used [45]. Studying small industrial programs

written in the 1980s, Gill and Kemerer found that code complexity correlates with software maintenance productivity [17]. Based on interviews and surveys with professional software developers, Besker and colleagues found that technical debt correlates negatively with developer productivity [3, 4].

We measured code quality and technical debt with 5 subjective factors from our survey:

- *Code Quality Satisfaction* (sat. with project code quality, sat. with dependency code quality)
- *Code Technical Debt* (project tech debt)
- *Dependency Technical Debt* (dependency tech debt)
- *Technical Debt Hindrance* (tech debt hindrance)

4.3.2 Infrastructure Tools & Support. The next category of potential drivers of productivity are issues relating to tools and infrastructure. Prior work showed that using "the best tools and practices" was the strongest correlate of individual productivity at Google, though not a significant correlate at two other companies [36]. Storey and colleagues also found that Microsoft developers' processes and tools correlated with individual productivity [47].

This category had 6 objective and 12 subjective measures:

- *Tools, infrastructure and service satisfaction* (sat. with infra & tools)
- *Tools and infrastructure choice* (choices of infra & tools)
- *Tools and infrastructure innovativeness* (innovation of infra & tools)
- *Tools and infrastructure ease* (ease of infra & tools)
- *Tools and infrastructure frustration* (frustration of infra & tools)
- *Developer stack change* (change of tool stack)
- *Internal documentation support* (doc. support)
- *Internal documentation hindrance* (doc. hindrance)
- *Build & test cycle hindrance* (build & test cycle hindrance)
- *Build latency satisfaction* (sat. with build latency)
- *50th and 90th percentile of build duration* (p50 build time, p90 build time)
- *% of long builds per week* (% of long builds)
- *50th and 90th percentile of test duration* (p50 test time, p90 test time)
- *% of long tests per week* (% of long tests)
- *Learning hindrance* (learning hindrance)
- *Migration hindrance* (migration hindrance)

4.3.3 Team Communication. The next category of drivers of productivity are issues relating to team communication. In a survey of knowledge workers, Hernaus and Mikulić found that social job characteristics (e.g. group cooperation) correlated with contextual job performance [23]. More specifically, in software engineering, Chatzoglou and Macaulay interviewed software developers, finding that most believed that communication among team members was very important to project success [9]. Studying communication networks quantitatively, Kidane and Gloor found that in the Eclipse project, a higher frequency of communication between developer correlated positively with performance and creativity [28].

To measure team communication in our study, we examined 9 objective measures and 1 subjective measure:

- *50th and 90th percentile of rounds of code review* (p50 code review rounds, p90 code review rounds)
- *50th and 90th percentile of total wait time of code review* (p50 code review wait time, p90 code review wait time)
- *50th and 90th percentile of code reviewers' organizational distances from author* (p50 review org distance, p90 review org distance)
- *50th and 90th percentile of code reviewers' physical distances from author* (p50 review physical distance, p90 review physical distance)
- *Physical distance from direct manager* (distance from manager)
- *Code review hindrance* (slow code review)

4.3.4 Goals and Priorities. Prior research suggests that changing goals and priorities correlate with software engineering outcomes. Surveying 365 software developers, The Standish Group found that changing requirements was a common stated reason for project failure [48]. Meyer and colleagues found that one of the top 5 most commonly mentioned reasons for a productive workday was having clear goals and requirements [34].

We measure this category with 1 subjective measure:

- *Priority shift* (priority shift)

4.3.5 Interruptions. Meyer and colleagues found that two of the top five most commonly mentioned reasons for a productive workday by 379 software developers was having no meetings and few interruptions [34]. Similarly, a prior survey of Google engineers showed that lack of interruptions and efficient meetings correlated with personal productivity, as did use of personal judgment [36].

We measure this category with 3 objective measures:

- *50th and 90th percentile of total time spent on incoming meetings per week* (p50 meeting time, p90 meeting time)
- *Total time spent on any meetings per week* (total meeting time)

4.3.6 Organizational and Process Factors. Finally, outside of software engineering, organizational and process factors correlate with a variety of work outcomes. For example, according to healthcare industry managers, reorganizations can result in workers' sense of powerlessness, inadequacy, and burnout [19]. Although not well-studied in software engineering, based on personal experience, DeMarco and Lister [11] and Armour [2] point to bureaucracy and reorganizations as leading to poor software engineering outcomes.

This category had 2 subjective and 3 objective measures:

- *Process hindrance* (complicated processes)
- *Organizational hindrance* (team & org change)
- *Number of times when engineers' direct manager changes but colleagues do not change* (reorg direct manager change)
- *Number of times when both an engineer's direct manager and colleagues change simultaneously* (non-reorg direct manager change)
- *Number of different primary teams the engineer has* (primary team change)

4.4 From Variables to Panel Data

Since the survey was sent out to the same cohort of engineers every three quarters, we have accumulated a panel data set with two observations in different points of time for each engineer. After joining each engineer's survey data with their logs data, we have complete panel data for 2139 engineers.

4.5 Modeling

Using the panel data set, we applied a quasi-experiment method of panel data analysis to analyze the relationship between engineers' perceived overall productivity and the independent variables. In this paper, we use a fixed-effect model to analyze panel data at the developer level. The model is

$$y_{it} = \alpha_i + \lambda_t + \beta D_{it} + \epsilon_{it} \quad (1)$$

where

- y_{it} is the dependent variable y, self-rated productivity for developer i at time t.
- α_i is unobserved engineer time-invariant effects, such as education and skills.
- λ_t is the engineer-independent time effect, such as company-wide policy changes and seasonalities at time t.
- $D_{it} = [D^1_{it}, D^2_{it}, \ldots, D^n_{it}]$ are observed productivity factors for developer i at time t.
- $\beta = [\beta^1, \beta^2, \ldots, \beta^n]$ are the causal effects of productivity factors D_{it} at time t.
- ϵ_{it} is the error term at time t.

To estimate the fixed-effect model, we differenced equation (1) between the two periods and have

$$\Delta y_{it} = \Delta\lambda_t + \beta \Delta D_{it} + \Delta\epsilon_{it} \quad (2)$$

where $\Delta\lambda_t = \gamma_0 + \gamma_1 T$. The Δ prefix denotes the change from one time period to the next. T is a categorical variable representing panels in different time periods, if we have more than one panel. Note that after differencing, α_i is cancelled out and $\Delta\lambda_t$ can be explicitly controlled by transforming it to a series of dummy variables. Therefore, factors in α_i and λ_t do not confound the results.

We then estimated equation (2) using Feasible Generalized Least Squares (FGLS); we chose FGLS to overcome heteroskedasticity, serial correlation between residuals, and for efficiency compared to Ordinary Least Square estimators. The parameters of interest are the β terms. The hypothesis we are testing is that $\beta = 0$ for all D_{it}. Except for binary variables and percentage variables, we transform D_{it} into $log(D_{it})$. The benefit of taking a natural log is to allow us to interpret estimates of regression coefficients (β terms) as an elasticity, where a percent change in a dependent variable can be interpreted as a percent change in an independent variable. This allows for both a uniform and intuitive interpretation of the effects across both logs-based and survey-based dependent variables.

To liberally capture causal relationships between productivity, we use a p-value cutoff of 0.1 to define "statistically significant" results. If the reader prefers a more stringent cutoff or using a false discovery correction, we facilitate this by reporting p-values.

Analysis code was written in R by the first author using the packages glmnet, randomForest, binom, car, and plm. All code was peer-reviewed using Google's standard code review process [43].

4.6 Multicollinearity

To check for multicollinearity among the independent variables, we calculated Variance Inflation Factor (VIF) scores on these metrics. We found some build latency metrics were highly correlated and thus may cause a multicollinearity problem. After consulting with experts in our build system, we removed three build latency metrics that had a VIF score above 3 (p50 build time, p50 test time, and % of long tests), a threshold recommended by Hair and colleagues [22]. The final list of 39 metrics all have VIF scores below 3.

4.7 Threats to Validity

Like all empirical studies, ours is imperfect. In this section, we describe threats to the validity of our study, broken down into content, construct, internal, and external validity threats.

4.7.1 Content. Although our study examines a variety of facets of productivity, it does not examine every single aspect of productivity or of factors that may influence productivity.

With respect to productivity itself, we measure it with a single survey question. On one hand, the question itself is worded broadly and our validation (Section 4.2) shows that it correlates with other objective measures of productivity. On the other hand, as evidenced by the fact that the correlation was imperfect, it is likely that our question did not capture some aspects of developer productivity. As one example, our question was only focused on productivity of an individual developer, yet productivity is often conceptualized from a team, group, or company perspective [16].

Likewise, our set of productivity factors – like code quality and build speed – are incomplete, largely because we used conveniently available and subjectively-selected metrics and because we reused an existing long-running survey. In comparison, prior work, which used a custom-built cross-sectional survey, found that two of the strongest correlates with individual productivity were job enthusiasm and teammates' support for new ideas [36]. Neither of these two productivity factors were explored in the present survey, demonstrating that our productivity factors are incomplete.

4.7.2 Construct. Our EngSat survey measures a variety of theoretical concepts, and the questions contained in it contain a range of construct validity. For instance, while we have demonstrated some amount of convergent validity of our productivity question, respondents to the question may have interpreted the word "productivity" differently – some may have interpreted it to refer only to the quick completion of work items, while others might take a more expansive view to include aspects such as quality. While we have tried to limit the impact of different interpretations of EngSat questions by piloting variations, gathering interpretive feedback, and refining wording iteratively, such issues are unavoidable threats.

Another specific threat to construct validity is inconsistent and ambiguous question wording. For instance, while respondents are advised at the beginning of the survey that they should report on experiences over the last 3 months, some questions (but not all) reinforce this scoping by beginning with "In the last three months…". As another example of inconsistency, while most questions ask only about experiences (which our models use to predict productivity), three questions ask about the relationship between experience and perceived productivity, such as "how much has technical debt…

Table 3: Metrics' relationship with self-rated productivity.

Metric	Effect size	p-value
Code Quality & Technical Debt		
sat. with project code quality	0.105	<0.001
sat. with dependency code quality	-0.013	0.505
project tech debt	0.078	<0.001
dependency tech debt	0.042	0.012
tech debt hindrance	-0.009	0.459
Infrastructure Tools & Support		
sat. with infra & tools	0.113	<0.001
choices of infra & tools	0.020	0.083
innovation of infra & tools	0.106	<0.001
ease of infra & tools	-0.018	0.352
frustration of infra & tools	0.002	0.952
change of tool stack	0.019	0.098
doc. support	-0.009	0.664
doc. hindrance	-0.005	0.715
build and test cycle hindrance	0.029	0.064
sat. with build latency	0.018	0.295
p90 build time	-0.024	0.019
p90 test time	-0.001	0.836
% of long builds	0.028	0.599
learning hindrance	0.038	0.006
migration hindrance	-0.001	0.929
Team Communication		
p50 code review rounds	0.007	0.081
p90 code review rounds	-0.014	0.058
p50 code review wait time	-0.0006	0.875
p90 code review wait time	0.0019	0.625
p50 review org distance	-0.0008	0.424
p90 review org distance	-0.0002	0.880
p50 review physical distance	0.0012	0.261
p90 review physical distance	0.0013	0.518
distance from manager	0.001	0.209
slow code review	0.051	0.004
Goals & Priorities		
priority shift	0.077	<0.001
Interruptions		
p50 meeting time	0.014	0.502
p90 meeting time	0.008	0.701
total meeting time	-0.009	0.692
Organizational Change and Process		
complicated processes	0.027	0.067
team and org change	0.032	0.023
reorg direct manager changes	-0.002	0.086
non-reorg direct manager change	-0.002	0.525
primary team change	0.014	0.086

hindered your productivity?". As an example of ambiguity, several questions ask about engineers' experiences with the project they work on, but respondents interpret for themselves what a "project" is and, if they work on multiple projects, which one to report on.

4.7.3 Internal. As we argue in this paper, our use of panel analysis helps draw stronger causal inferences than those that can be drawn from cross-sectional data. However, the most significant caveat to our ability to draw causal inferences is time variant effects. In contrast to time invariant effects (e.g., prior education and demographics), time variant effects may vary over the study period. For instance, in our running example, if Aruj lost a mentor and Rusla gained a mentor between the two surveys, our analysis could not rule out mentorship as a cause of increased productivity or code quality. Thus, our analysis assumes that effects on individual engineers are time invariant. Violations of this assumption threaten the internal validity of our study.

Another internal threat to the validity of our study is participants who chose not to answer some or all questions in the survey. While our analysis of non-response bias (Section 4.1.2) showed that two survey questions were robust to non-response among several dimensions like level and tenure, non-response is still a threat. For one, respondents and non-respondents might differ systematically on some unmeasured or dimension, such as how frequently they get feedback from peers. Likewise, respondents who choose not to answer a question will be wholly excluded from our analysis, yet such participants might differ systematically from those who answered every question.

Another threat to internal validity is that we analyzed data for only two panels per engineer. More panels per engineer would increase the robustness of our results.

4.7.4 External. As the title of this paper suggests, our study was conducted only at Google and generalizability of our results beyond that context is limited. Google is a large, US-headquartered, multinational, and software-centric company where engineers work on largely server and mobile code, with uniform development tooling, and in a monolithic repository. Likewise, during the study period Google developers mostly worked from open offices, before the global COVID19 pandemic when many developers shifted to remote or hybrid work. While results would vary if this study were replicated in other organizations, contexts that resemble ours are most likely to yield similar results.

5 PANEL ANALYSIS: RESULTS

5.1 Factors Causally Linked to Productivity

Panel data analysis suggested that 16 out of the 39 metrics have a statistically significant causal relationship with perceived overall productivity, as listed in Table 3. The overall adjusted R-squared value for the model was 0.1019. In Table 1, the Effect size should be read as a percent change in the dependent variable is associated with that percent change in the independent variable. For instance, for code quality, a 100% change in project code quality (from "Very dissatisfied" to "Very satisfied" to quality) is associated with a 10.5% increase in self-reported productivity. To summarize Table 3:

- For code quality, we found that perceived productivity is causally related to satisfaction with project code quality but not causally related to satisfaction with code quality in dependencies. For technical debt, we found perceived productivity is causally related to perceived technical debt both within projects and in their dependencies.

- For infrastructure, several factors closely related to internal infrastructure and tools showed a significant causal relationship with perceived productivity:
  - Engineers who reported their tools and infrastructure were not innovative were more likely to report lower productivity.
  - Engineers who reported the number of choices were either too few or too many were likely to report lower productivity. We further tested whether one of the two ("too few" or "too many") matters but not the other, by replacing this variable with two binary variables, one representing the case of "too few" choices and the other representing the case of "too many" choices. The results suggest that both cases are causally related to perceived productivity.
  - Engineers who reported that the pace of changes in the developer tool stack was too fast or too slow were likely to report lower productivity. Similarly, we tested the two cases, "too fast" or "too slow", separately by replacing this variable with two binary variables, one representing the case of "too fast" and the other representing the case of "too slow". Results suggested both cases matter for perceived productivity.
  - Engineers who were hindered by learning a new platform, framework, technology, or infrastructure were likely to report lower productivity.
  - Engineers who had longer build times or reported being hindered by their build & test cycle were more likely to report lower productivity.
- For team communication, a metric related to code review was significantly causally related with perceived productivity. Engineers who had more rounds of reviews per code review or reported being hindered by slow code review processes were likely to report lower productivity.
- For goals and priorities, engineers hindered by shifting project priorities were likely to report lower productivity.
- Organizational factors were linked to perceived productivity:
  - Engineers who had more changes of direct managers were more likely to report lower productivity.
  - Engineers who reported being hindered for team and organizational reasons, or by complicated processes were more likely to report lower productivity.

5.2 Quadrant Chart

To visualize these factors in terms of their relative effect size and statistical significance, we plot them in a quadrant chart (Figure 2). The chart excludes factors whose p-value is greater than 0.1. The factors have various scales from satisfaction score to time duration, so to make their effect size comparable, we standardized metrics by subtracting each data point by its mean and dividing it by its standard deviation. The x axis is the absolute value of the standardized effect size. The y axis is p-values.

The top five factors in terms of relative effect size are satisfaction with project code quality, hindrance of shifting priorities, technical debt in projects, innovation of infrastructure, and tools and overall satisfaction with infrastructure and tools.

6 LAGGED PANEL ANALYSIS: METHODS

The panel data analysis we conducted so far suggests satisfaction with code quality within projects is the strongest productivity factor among the 39 we studied, based on standardized effect size and p-value.

However, because the observed changes in factors coincided during the same time period, such conventional panel data analysis can tell which factors are causally related to overall productivity, but it does not tell us the direction of the causality.

So, does better code quality cause increasing productivity, or does increasing productivity cause improved code quality? Both linkages are theoretically plausible: on one hand, code quality might increase productivity because higher code quality may make it easier and faster to add new features; on the other hand, high productivity might increase quality code because engineers have free time to spend on quality improvement.

To verify the direction of the causal relationship between project code quality and productivity, we conducted another panel data analysis using lagged panel data. In this analysis, we focus only on the causal relationship between code quality and productivity. Although such an analysis is possible for other factors, it is nonetheless laborious, as we shall see shortly. Thus, we focus our lagged analysis on only these two variables, which had the strongest causal relationship in our prior analysis.

In short, we verified the direction of the linkage between project code quality and productivity by checking if the change in one factor is associated with the change in the other factor in the following period. The idea is that if project code quality affects productivity, we expect to see that changes in project code quality during time T-1 are associated with changes in productivity during time T. Since self-reported productivity is not available for two consecutive quarters (since each respondent is sampled only once every three quarters), we switch to logs-based metrics to measure productivity. Complementing our prior analysis based on self-ratings with a logs-based one has the additional benefit of increasing the robustness of our results.

More formally, we tested two competing hypotheses, Hypothesis QaP (Quality affects Productivity) and PaQ (Productivity affects Quality). Hypothesis QaP is that the changes in project code quality during time T-1 are associated with changes in productivity during time T. This implies improvements in project code quality lead to better productivity. Hypothesis PaQ is that changes in productivity in time T-1 are associated with changes in project code quality in time T. This implies better productivity leads to an improvement in project code quality.

Hypothesis QaP: Changes in code quality during time T-1 are correlated with changes in productivity during time T. The statistical model is

$$\Delta P_{it} = \alpha + \beta \Delta Q_{it-1} + \Delta \epsilon_{it} \quad (4)$$

where ΔQ_{it-1} is the change in code quality at time t-1 and ΔP_{it} is the following change in logs-based productivity metrics at time t. Given the available data, we use the difference between Q3 2018 and Q2 2019 to measure ΔQ_{it-1} and the difference between Q3 2018 and Q3 2019 to measure ΔP_{it}.

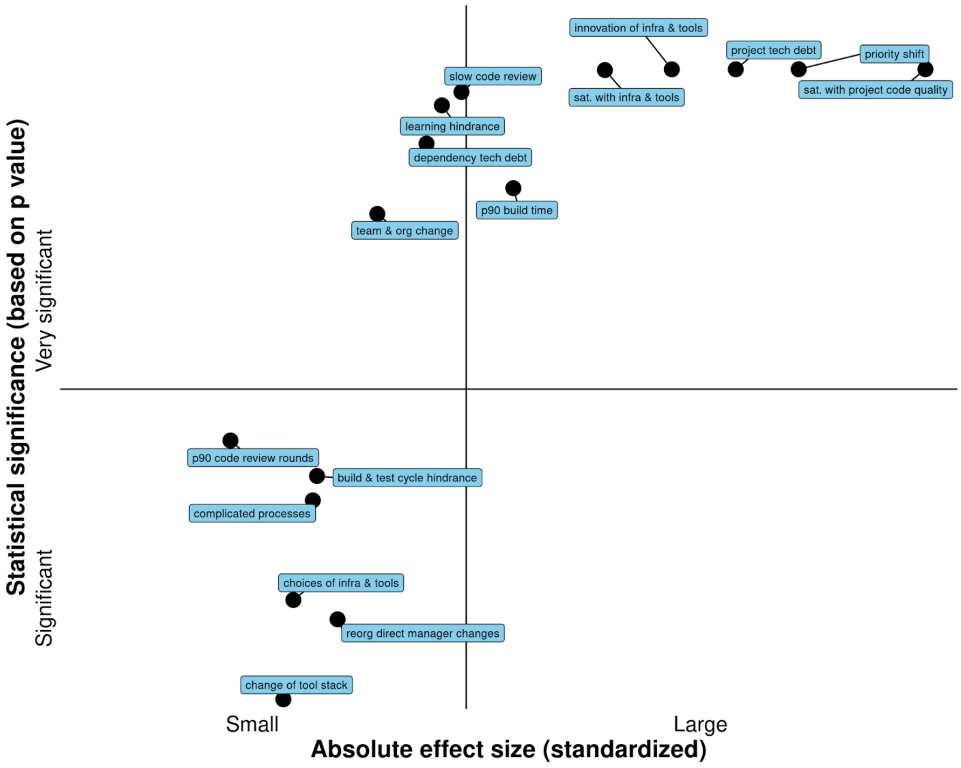

Figure 2: Quadrant of productivity factors in effect size and statistical significance

Hypothesis PaQ: Changes in productivity in time T-1 are correlated with changes in code quality in time T. The statistical model is

$$\Delta Q_{it} = \alpha + \beta \Delta P_{it-1} + \Delta \epsilon_{it} \quad (5)$$

where ΔP_{it-1} is the change in logs-based productivity at time t-1 and ΔQ_{it} is the following change in code quality at time t. Given the availability of data, we use the difference between Q3 2018 and Q2 2019 to measure ΔP_{it-1} and the difference between Q3 2018 and Q1 2019 to measure ΔQ_{it}.

For this analysis, we had full lagged panel data for 3389 engineers.

7 LAGGED PANEL ANALYSIS: RESULTS

Our results support hypothesis QaP but not hypothesis PaQ. We found that a 100% increase of satisfaction rating with project code quality (i.e. going from a rating of 'Very dissatisfied' to 'Very satisfied') at time T-1 was associated with a 10% decrease of median active coding time per CL, a 12% decrease of median wall-clock time from creating to mailing a CL, and a 22% decrease of median wall-clock time from submitting to deploying a CL at time T. On the other hand, we did not find any evidence to support hypothesis PaQ; changes in satisfaction with project code quality in time T were not associated with any of the productivity metrics in time T-1. See Appendix for a table containing this data and descriptions of each variable. Therefore, returning to our research question, we conclude that changes in satisfaction with project code quality cause changes in perceived overall productivity.

8 DISCUSSION

Our findings provide practical guidance for organizations trying to improve individual developer productivity by providing a list of amenable factors that are causally linked to productivity. Specifically, our panel analysis shows that these factors are: code quality, technical debt, infrastructure tools and support, team communication, goals and priorities, and organizational change and process. Our quadrant chart shown in Figure 2, which we originally created for an executive stakeholder audience within Google, allows practitioners to choose highly impactful productivity factors to act on. Factors at the top of the chart are those with high statistical significance (and low standard error), so practitioners can read those as the most consistent productivity factors. Factors on the right are the ones with the largest standardized effect size, so these supply the "biggest bang for the buck". Taken together, the factors in the upper right quadrant are the ones most promising to improve productivity at Google. For instance, giving teams time to improve code quality, reduce technical debt, and stabilize priorities would be good candidate initiatives for improving individual developer productivity.

We found that several factors did not have a statistically significant relationship with perceived productivity, notably:

- For documentation, perceived productivity was not causally linked to reported poor or missing documentation (*doc. hindrance*) or the frequency of documentation meeting needs (*doc. support*). This is surprising, given that GitHub's 2017 survey of 5,500 developers found that "incomplete or confusing documentation" was the most commonly encountered problem in open source [18]. GitHub's findings are consistent with findings at Microsoft [47] and at Google– EngSat respondents often report "poor or missing documentation" as one of the top three hindrances to their own productivity. However, the results in this paper suggest that there is no causal relationship between developer productivity and documentation, despite developers' reports that it is important to them. One way to explain this finding is that documentation may not impact productivity, but it may yet have other positive benefits, such as to "create inclusive communities" [18].
- For meetings, we found that perceived productivity was not causally linked to time spent on either incoming meetings(*p50 meeting time*, *p90 meeting time*) or all types of meetings (*total meeting time*). This is also surprising, given that prior research found in a survey of Microsoft engineers that meetings were the most unproductive activity for engineers [34]. The contradictory results could be explained by differences between the studies: our panel analysis enables causal reasoning (vs correlational), more engineers were represented in our dataset (2139 vs 379), and we used objective meeting data from engineers' calendars (vs. self-reports).
- For physical and organizational distances, perceived productivity was not causally linked to physical distance from direct manager (*distance from manager*), or physical (*p50 review physical distance, p90 review physical distance*) or organizational distances from code reviewers(*p50 review org distance, p90 review org distance*). This is in contrast to Ramasubbu and colleagues' cross-sectional study, which found that "as firms distribute their software development across longer distance (and time zones) they benefit from improved project level productivity" [41]. As with the prior differences, explanatory factors may include differences in organization and a methodology: individual productivity versus organizational productivity, single company versus multiple companies, and panel versus cross-sectional analysis.

As we mentioned, a threat to these results is the threat of reverse causality – the statistics do not tell us whether each factor causes productivity changes or vice versa. We mitigated this threat for code quality using lagged panel analysis, providing compelling evidence that high code quality increases individual developers' productivity.

Within Google, our results have driven organizational change around code quality and technical debt as a way to improve developer productivity:

- Since its creation in May 2019, a version of this report has been viewed by more than 1000 unique Google employees with more than 500 comments.
- EngSat results helped motivate two code quality conferences for Google engineers with 4,000 internal attendees and more than 15,000 views of live and on-demand talks.
- The research motivated the creation of two initiatives – a Technical Debt Maturity Model (akin to the Capability Maturity Model [38]) and Technical Debt Management Framework – to help teams improve technical debt assessment and management.
- Several teams and organizations set Objectives and Key Results (OKRs) [12] to improve technical debt in their workgroups.
- Google introduced "The Healthys", an award where teams submit a two page explanation of a code quality improvement initiative they've performed. Using an academic reviewing model, outside engineers evaluated the impact of nearly 350 submissions across the company. Accomplishments include more than a million lines of code deleted. In a survey sent to award recipients, of 173 respondents, most respondents reported that they mentioned the award in the self-evaluation portion of their performance evaluation (82%) and that there was at least a slight improvement in how code health work is viewed by their team (68%) and management (60%).

Although difficult to ascribe specifically to this research and the above initiatives that it has influenced, EngSat has revealed several encouraging trends between when the report was released internally in the second quarter of 2019 and the first quarter of 2021: The proportion of engineers feeling "not at all hindered" by technical debt has increased by 27%. The proportion of engineers feeling satisfied with code quality has increased by about 22%. The proportion of engineers feeling highly productive at work has increased by about 18%.

9 CONCLUSION

Prior research has made significant progress in improving our understanding of what correlates with developer productivity. In this paper, we've advanced that research by leveraging time series data to run panel analyses, enabling stronger causal inference than was possible in prior studies. Our panel analysis suggests that code quality, technical debt, infrastructure tools and support, team communication, goals and priorities, and organizational change and process are causally linked to developer productivity at Google. Furthermore, our lagged panel analysis provides evidence that improvements in code quality cause improvements in individual productivity. While our analysis is imperfect – in particular, it is only one company and uses limited measurements – it nonetheless can help engineering organizations make informed decisions about improving individual developer productivity.

ACKNOWLEDGMENT

Thanks to Google employees for contributing their EngSat and logs data to this study, as well as the teams responsible for building the infrastructure we leverage in this paper. Thanks in particular to Adam Brown, Michael Brundage, Yuangfang Cai, Alison Chang, Sarah D'Angelo, Daniel Dressler, Ben Holtz, Matt Jorde, Kurt Kluever, Justin Purl, Gina Roldan, Alvaro Sanchez Canudas, Jason Schwarz, Simone Styr, Fred Wiesinger, and anonymous reviewers.

REFERENCES

[1] Joshua D. Angrist and Jörn-Steffen Pischke. 2008. *Mostly harmless econometrics: An empiricist's companion.* Princeton University Press.

[2] Phillip Armour. 2003. The Reorg Cycle. *Commun. ACM* 46, 2 (2003), 19.

[3] Terese Besker, Hadi Ghanbari, Antonio Martini, and Jan Bosch. 2020. The influence of Technical Debt on software developer morale. *Journal of Systems and Software* 167 (2020), 110586. https://doi.org/10.1016/j.jss.2020.110586

[4] Terese Besker, Antonio Martini, and Jan Bosch. 2019. Software developer productivity loss due to technical debt—a replication and extension study examining developers' development work. *Journal of Systems and Software* 156 (2019), 41–61. https://doi.org/10.1016/j.jss.2019.06.004

[5] Larissa Braz, Enrico Fregnan, Gül Çalikli, and Alberto Bacchelli. 2021. Why Don't Developers Detect Improper Input Validation?'; DROP TABLE Papers;–. In *International Conference on Software Engineering*. IEEE, 499–511. https://doi.org/10.1109/ICSE43902.2021.00054

[6] K.H. Brodersen, F. Gallusser, J. Koehler, N. Remy, and S.L. Scott. 2015. Inferring causal impact using Bayesian structural time-series models. *The Annals of Applied Statistics* 9, 1 (2015), 247–274. https://doi.org/10.1214/14-AOAS788

[7] Kevin D Carlson and Andrew O Herdman. 2012. Understanding the impact of convergent validity on research results. *Organizational Research Methods* 15, 1 (2012), 17–32. https://doi.org/10.1177/1094428110392383

[8] Nancy Cartwright. 2007. Are RCTs the gold standard? *BioSocieties* 2, 1 (2007), 11–20. https://doi.org/10.1017/S1745855207005029

[9] Prodromos D. Chatzoglou and Linda A. Macaulay. 1997. The importance of human factors in planning the requirements capture stage of a project. *International Journal of Project Management* 15, 1 (1997), 39–53. https://doi.org/10.1016/S0263-7863(96)00038-5

[10] Bradford Clark, Sunita Devnani-Chulani, and Barry Boehm. 1998. Calibrating the COCOMO II post-architecture model. In *Proceedings of the International Conference on Software Engineering*. IEEE, 477–48. https://doi.org/10.1109/ICSE.1998.671610

[11] Tom DeMarco and Tim Lister. 2013. *Peopleware: productive projects and teams.* Addison-Wesley.

[12] John Doerr. 2018. *Measure what matters: How Google, Bono, and the Gates Foundation rock the world with OKRs.* Penguin.

[13] Davide Falessi, Natalia Juristo, Claes Wohlin, Burak Turhan, Jürgen Münch, Andreas Jedlitschka, and Markku Oivo. 2018. Empirical software engineering experts on the use of students and professionals in experiments. *Empirical Software Engineering* 23, 1 (2018), 452–489. https://doi.org/10.1007/s10664-017-9523-3

[14] Petra Filkuková and Magne Jørgensen. 2020. How to pose for a professional photo: The effect of three facial expressions on perception of competence of a software developer. *Australian Journal of Psychology* 72, 3 (2020), 257–266. https://doi.org/10.1111/ajpy.12285

[15] Denae Ford, Margaret-Anne Storey, Thomas Zimmermann, Christian Bird, Sonia Jaffe, Chandra Maddila, Jenna L. Butler, Brian Houck, and Nachiappan Nagappan. 2021. A Tale of Two Cities: Software Developers Working from Home during the COVID-19 Pandemic. *ACM Trans. Softw. Eng. Methodol.* 31, 2, Article 27 (dec 2021), 37 pages. https://doi.org/10.1145/3487567

[16] Nicole Forsgren, Margaret-Anne Storey, Chandra Maddila, Thomas Zimmermann, Brian Houck, and Jenna Butler. 2021. The SPACE of Developer Productivity: There's more to it than you think. *Queue* 19, 1 (2021), 20–48. https://doi.org/10.1145/3454122.3454124

[17] Geoffrey K. Gill and Chris F. Kemerer. 1991. Cyclomatic complexity density and software maintenance productivity. *IEEE transactions on software engineering* 17, 12 (1991), 1284. https://doi.org/10.1109/32.106988

[18] GitHub. 2017. Open Source Survey. https://opensourcesurvey.org/2017/

[19] Ann-Louise Glasberg, Astrid Norberg, and Anna Söderberg. 2007. Sources of burnout among healthcare employees as perceived by managers. *Journal of Advanced nursing* 60, 1 (2007), 10–19. https://doi.org/10.1111/j.1365-2648.2007.04370.x

[20] C. W. Granger. 1969. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica: Journal of the Econometric Society* (1969), 424–438. https://doi.org/10.2307/1912791

[21] Shenyang Guo and Mark W. Fraser. 2014. *Propensity score analysis: Statistical methods and applications.* Vol. 11. SAGE publications.

[22] Joseph F Hair, Jeffrey J Risher, Marko Sarstedt, and Christian M Ringle. 2019. When to use and how to report the results of PLS-SEM. *European Business Review* (2019). https://doi.org/10.1108/EBR-11-2018-0203

[23] Tomislav Hernaus and Josip Mikulić. 2014. Work characteristics and work performance of knowledge workers. *EuroMed Journal of Business* (2014). https://doi.org/10.1108/EMJB-11-2013-0054

[24] Cheng Hsiao. 2007. Panel data analysis—advantages and challenges. *TEST* 16, 1 (2007), 1–22. https://doi.org/10.1007/s11749-007-0046-x

[25] Cheng Hsiao. 2022. *Analysis of panel data.* Cambridge University Press.

[26] Mazhar Islam, Jacob Miller, and Haemin Dennis Park. 2017. But what will it cost me? How do private costs of participation affect open source software projects? *Research Policy* 46, 6 (2017), 1062–1070. https://doi.org/10.1016/j.respol.2017.05.005

[27] Ciera Jaspan, Matt Jorde, Carolyn Egelman, Collin Green, Ben Holtz, Edward Smith, Maggie Hodges, Andrea Knight, Liz Kammer, Jill Dicker, et al. 2020. Enabling the Study of Software Development Behavior With Cross-Tool Logs. *IEEE Software* 37, 6 (2020), 44–51. https://doi.org/10.1109/MS.2020.3014573

[28] Yared H. Kidane and Peter A. Gloor. 2007. Correlating temporal communication patterns of the Eclipse open source community with performance and creativity. *Computational and mathematical organization theory* 13, 1 (2007), 17–27. https://doi.org/10.1007/s10588-006-9006-3

[29] Amy J. Ko. 2019. Individual, Team, Organization, and Market: Four Lenses of Productivity. In *Rethinking Productivity in Software Engineering*. Springer, 49–55. https://doi.org/10.1007/978-1-4842-4221-6_6

[30] Amy J. Ko and Brad A. Myers. 2008. Debugging Reinvented: Asking and Answering Why and Why Not Questions about Program Behavior. In *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*. Association for Computing Machinery, New York, NY, USA, 301–310. https://doi.org/10.1145/1368088.1368130

[31] Max Lillack, Stefan Stanciulescu, Wilhelm Hedman, Thorsten Berger, and Andrzej Wąsowski. 2019. Intention-Based Integration of Software Variants. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 831–842. https://doi.org/10.1109/ICSE.2019.00090

[32] William Martin, Federica Sarro, and Mark Harman. 2016. Causal impact analysis for app releases in Google Play. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 435–446. https://doi.org/10.1145/2950290.2950320

[33] Katrina D. Maxwell, Luk Van Wassenhove, and Soumitra Dutta. 1996. Software development productivity of European space, military, and industrial applications. *IEEE Transactions on Software Engineering* 22, 10 (1996), 706–718. https://doi.org/10.1109/32.544349

[34] André N Meyer, Thomas Fritz, Gail C Murphy, and Thomas Zimmermann. 2014. Software developers' perceptions of productivity. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 19–29. https://doi.org/10.1145/2635868.2635892

[35] Emerson Murphy-Hill and Andrew P. Black. 2008. Breaking the Barriers to Successful Refactoring: Observations and Tools for Extract Method. In *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*. Association for Computing Machinery, New York, NY, USA, 421–430. https://doi.org/10.1145/1368088.1368146

[36] Emerson Murphy-Hill, Ciera Jaspan, Caitlin Sadowski, David Shepherd, Michael Phillips, Collin Winter, Andrea Knight, Edward Smith, and Matthew Jorde. 2021. What Predicts Software Developers' Productivity? *IEEE Transactions on Software Engineering* 47, 3 (2021), 582–594. https://doi.org/10.1109/TSE.2019.2900308

[37] Edson Oliveira, Eduardo Fernandes, Igor Steinmacher, Marco Cristo, Tayana Conte, and Alessandro Garcia. 2020. Code and commit metrics of developer productivity: a study on team leaders perceptions. *Empirical Software Engineering* 25, 4 (2020), 2519–2549. https://doi.org/10.1007/s10664-020-09820-z

[38] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber. 1993. Capability maturity model, version 1.1. *IEEE Software* 10, 4 (1993), 18–27. https://doi.org/10.1109/52.219617

[39] Kai Petersen. 2011. Measuring and predicting software productivity: A systematic map and review. *Information and Software Technology* 53, 4 (2011), 317–343. https://doi.org/10.1016/j.infsof.2010.12.001 Special section: Software Engineering track of the 24th Annual Symposium on Applied Computing.

[40] Huilian Sophie Qiu, Alexander Nolte, Anita Brown, Alexander Serebrenik, and Bogdan Vasilescu. 2019. Going Farther Together: The Impact of Social Capital on Sustained Participation in Open Source. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 688–699. https://doi.org/10.1109/ICSE.2019.00078

[41] Narayan Ramasubbu, Marcelo Cataldo, Rajesh Krishna Balan, and James D. Herbsleb. 2011. Configuring global software teams: a multi-company analysis of project productivity, quality, and profits. In *2011 33rd International Conference on Software Engineering (ICSE)*. 261–270. https://doi.org/10.1145/1985793.1985830

[42] Simone Romano, Davide Fucci, Maria Teresa Baldassarre, Danilo Caivano, and Giuseppe Scanniello. 2019. An empirical assessment on affective reactions of novice developers when applying test-driven development. In *International Conference on Product-Focused Software Process Improvement*. Springer, 3–19. https://doi.org/10.1007/978-3-030-35333-9_1

[43] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. 181–190. https://doi.org/10.1145/3183519.3183525

[44] Iflaah Salman, Ayse Tosun Misirli, and Natalia Juristo. 2015. Are students representatives of professionals in software engineering experiments?. In *International Conference on Software Engineering*, Vol. 1. IEEE, 666–676. https://doi.org/10.1109/ICSE.2015.82

[45] Andrea Schankin, Annika Berger, Daniel V. Holt, Johannes C. Hofmeister, Till Riedel, and Michael Beigl. 2018. Descriptive Compound Identifier Names Improve Source Code Comprehension. In *Proceedings of the 26th Conference on Program*

Comprehension (ICPC '18). Association for Computing Machinery, New York, NY, USA, 31–40. https://doi.org/10.1145/3196321.3196332

[46] Dag I.K. Sjoberg, Bente Anda, Erik Arisholm, Tore Dyba, Magne Jorgensen, Amela Karahasanovic, Espen Frimann Koren, and Marek Vokác. 2002. Conducting realistic experiments in software engineering. In *Proceedings International Symposium on Empirical Software Engineering*. 17–26. https://doi.org/10.1109/ISESE.2002.1166921

[47] Margaret-Anne Storey, Thomas Zimmermann, Christian Bird, Jacek Czerwonka, Brendan Murphy, and Eirini Kalliamvakou. 2021. Towards a Theory of Software Developer Job Satisfaction and Perceived Productivity. *IEEE Transactions on Software Engineering* 47, 10 (2021), 2125–2142. https://doi.org/10.1109/TSE.2019.2944354

[48] The Standish Group. 1995. The CHAOS report.

[49] Ayse Tosun, Oscar Dieste, Davide Fucci, Sira Vegas, Burak Turhan, Hakan Erdogmus, Adrian Santos, Markku Oivo, Kimmo Toro, Janne Jarvinen, and Natalia Juristo. 2017. An industry experiment on the effects of test-driven development on external quality and productivity. *Empirical Software Engineering* 22, 6 (2017), 2763–2805. https://doi.org/10.1007/s10664-016-9490-0

[50] Stefan Wagner and Florian Deissenboeck. 2019. Defining Productivity in Software Engineering. In *Rethinking Productivity in Software Engineering*, Caitlin Sadowski and Thomas Zimmermann (Eds.). Apress, Berkeley, CA, 29–38. https://doi.org/10.1007/978-1-4842-4221-6_4

[51] Zhendong Wang, Yi Wang, and David Redmiles. 2018. Competence-confidence gap: A threat to female developers' contribution on Github. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS*. IEEE, 81–90. https://doi.org/10.1145/3183428.3183437