What Predicts Software Developers' Productivity?

Emerson Murphy-Hill, Ciera Jaspan, Caitlin Sadowski, David Shepherd, Michael Phillips, Collin Winter, Andrea Knight, Edward Smith and Matthew Jorde

Abstract—Organizations have a variety of options to help their software developers become their most productive selves, from modifying office layouts, to investing in better tools, to cleaning up the source code. But which options will have the biggest impact? Drawing from the literature in software engineering and industrial/organizational psychology to identify factors that correlate with productivity, we designed a survey that asked 622 developers across 3 companies about these productivity factors and about self-rated productivity. Our results suggest that the factors that most strongly correlate with self-rated productivity were non-technical factors, such as job enthusiasm, peer support for new ideas, and receiving useful feedback about job performance. Compared to other knowledge workers, our results also suggest that software developers' self-rated productivity is more strongly related to task variety and ability to work remotely.

Index Terms—Productivity factors, software engineers, knowledge workers

1 Introduction

Improving productivity of software developers is important. By definition, developers who have completed their tasks can spend their freed-up time on other useful tasks, such as implementing new features or on new verification and validation activities. But what causes developers to be more productive?

Organizations such as ours demand empirical guidance on which factors to try to manipulate in order to best improve productivity. For example, should an individual developer (1a) spend time seeking out the best tools and practices, or (1b) shut down email notifications during the day?[1] Should a manager (2a) invest in refactoring to reduce code complexity, or (2b) give developers more autonomy over their work? Should executives (3a) invest in better software development tools, or (3b) should they invest in less distracting office space? In an ideal world, we would invest in a variety of productivity-improving factors, but time and money are limited, so we must make selective investments.

This paper contributes the broadest study to date to investigate what predicts software developers' productivity. As we detail in Section 3.1, productivity can be measured objectively (e.g. lines of code written per month) or subjectively (e.g. a developer's self-assessment); while neither is universally preferable, in this paper we seek to achieve breadth by measuring productivity subjectively to answer three research questions:

1) What factors most strongly predict developers' self-rated productivity?
2) How do these factors differ across companies?
3) What predicts developer self-rated productivity, in particular, compared to other knowledge workers?

To answer the first research question, we created a survey and deployed it at a large software company. To answer the second question, which helps establish to what extent our results generalize and do not generalize, we adapted and deployed the survey at two other companies in different industries. To answer the third question, which helps explain what differentiates software developers from other groups, we adapted and deployed the survey with non-developer knowledge workers, and compared the results to that of software developers.

Our results suggest that the factors that strongly correlate with self-rated productivity across the companies we studied were job enthusiasm, peer support for new ideas, and receiving useful feedback about job performance. Compared to other knowledge workers, our results also suggest that software developers' self-rated productivity is more strongly related to task variety and ability to work remotely. Our results can be used by companies to prioritize productivity initiatives, such as by increasing job enthusiasm through recognition of individuals' work (Section 4.7).

We next describe the three companies we studied in Section 2, describe and justify the methodology we used in Section 3, explain and discuss our results in Section 4, and finally describe related work in the area in Section 5.

2 Companies Studied

Before discussing our methodology, we next present descriptions of each company we studied. The descriptions are summarized in Table 1.

2.1 Google

Google has tens of thousands of developers spread across 40 offices worldwide. This software-centric company values colocating teams, and offices typically have open-plan layouts where developers on the same team work in close proximity. Google is relatively young (founded in the late 90s), and has a relatively flat organizational structure where individual developers have a lot of autonomy. The promotion process incorporates peer review feedback, and developers do not have to move into management roles to advance. Developers are free to schedule their time, and calendars are uniformly visible across the company. The software development process used

1. To guard against confirmation bias [1], we encourage the reader to circle (a) or (b) for each of the three options that they would recommend investing in. Readers who can confidently answer each question correctly can make a reasonable claim that the results of our study are predictable or unsurprising. Answers provided in Section 6.

	Google	ABB	National Instruments
Company Size	Large	Large	Small
Office layout	Open offices	Open and closed offices	Open offices
Tooling style	Mostly uniform developer tooling	Similar tooling	Flexible tooling
Development type	Mostly server and mobile code	Mix of web, embedded, desktop	Mostly embedded and desktop
Code repository	Monolithic repository	Separate repositories	Monolithic repository
Focus	Software-centric	Engineering conglomerate	Software/hardware-centric

TABLE 1: Profiles of the three companies studied

by developers at Google (e.g. agile) is flexible and typically determined on a per-team basis.

Google values openness; most developers at Google work in a shared monolithic codebase, and developers are empowered to change code in projects they don't own. There is a strong testing and code review culture; committed code has been reviewed by another developer and typically includes tests. Most developers at Google work on server code: code is typically released frequently, and it is relatively easy to push bug fixes. The set of developer tools used is largely uniform (with the exception of editors) and developed in-house, including the code review tool and the continuous integration and release infrastructure.

2.2 ABB

ABB has well over 100,000 employees worldwide. Because it is an engineering conglomerate, it employees a wide variety of workers. It has approximately 4,000 traditional software developers and over 10,000 application developers who program industrial systems using a variety of industry-specific visual and textual languages. Additionally, to run its large IT infrastructure, it has a significant staff whose duties include scripting and lightweight coding.

While ABB has acquired and incorporated smaller companies, it has had a central organization dedicated to ensuring a uniform software development processes. Thus, while variations occur across departments, much of the overall tooling and underlying approach to software development are similar. Similarly, while small variations occur across departments, most career development tracks are quite similar, progressing from junior to senior developers on the technical track and from group leader to department and finally center manager on the management track.

2.3 National Instruments

National Instruments was founded in the 1970s. Most software development takes place in four international research and development offices. Calendars are globally visible, and anyone can schedule meetings with anyone else.

Business commitments drive development activities. Developers don't have the autonomy to determine their project scope, but have input into specific tasks or features to work on. The majority of developers work in a shared monolithic codebase with clearly defined ownership between logical areas. Code submissions are expected to have received approval by a "code owner" before being allowed in. Code review from technical leads is also expected; this policy is not enforced but is widely followed.

There is a lot of freedom in terms of developer tools; developers are not consolidated onto common tooling unless it has immediate benefit. For example, the IDE used is heavily dependent on the kind of work being done, there is a mix of in-house tooling for build/test, and different portions of the company have standardized on different source control systems or code review tools. Software updates are typically released on a quarterly to yearly basis, with the exception of rare ad-hoc critical patch releases.

3 Methodology

Our goal is to comparatively understand the factors that predict software developers' productivity. To reach this goal, we designed a survey, consisting of a set of productivity questions, a set of productivity factors, and a set of demographic variables.

3.1 Self-Rated Productivity

The first step is to define how we will measure productivity. Ramírez and Nembhard provide a taxonomy of techniques used in the literature to measure productivity, including function point analysis, self-ratings, peer evaluations, outcome-input ratio, and professional time utilization [2]. These techniques can largely be divided into objective measurements (like lines of code written per week) and subjective measurements (like self-ratings and peer evaluations).

Neither objective nor subjective productivity measurements are universally preferable; both have drawbacks. Objective measurements suffer from inflexibility and gaming. For instance, consider lines of code written per week. A productive developer can write a one-line fix to a hard-to-find bug. At the same time, an unproductive developer can easily inflate the number of lines of code he writes. On the other hand, subjective measurements can suffer from inaccuracies due to cognitive biases. Consider peer evaluations. A productive developer may be disliked by her peers, who may in turn give her poor evaluations even if they try to remain objective.

Similar to Meyer and colleagues' recent investigation into software developer productivity [3], we have chosen to use survey questions as a subjective measurement of productivity, for two main reasons. First, as Ramírez and Nembhard note, surveys are a "straightforward and commonly used method to measure [knowledge worker] productivity" [2]. Second, surveys are flexible enough to allow responses from developers in a variety of roles and to allow the respondent to incorporate a variety of information into a self-rating of productivity.

We asked respondents about their productivity by rating their agreement with the following statement:

I regularly reach a high level of productivity.

Through this statement, we aimed to measure productivity in the broadest way possible. We began by designing eight

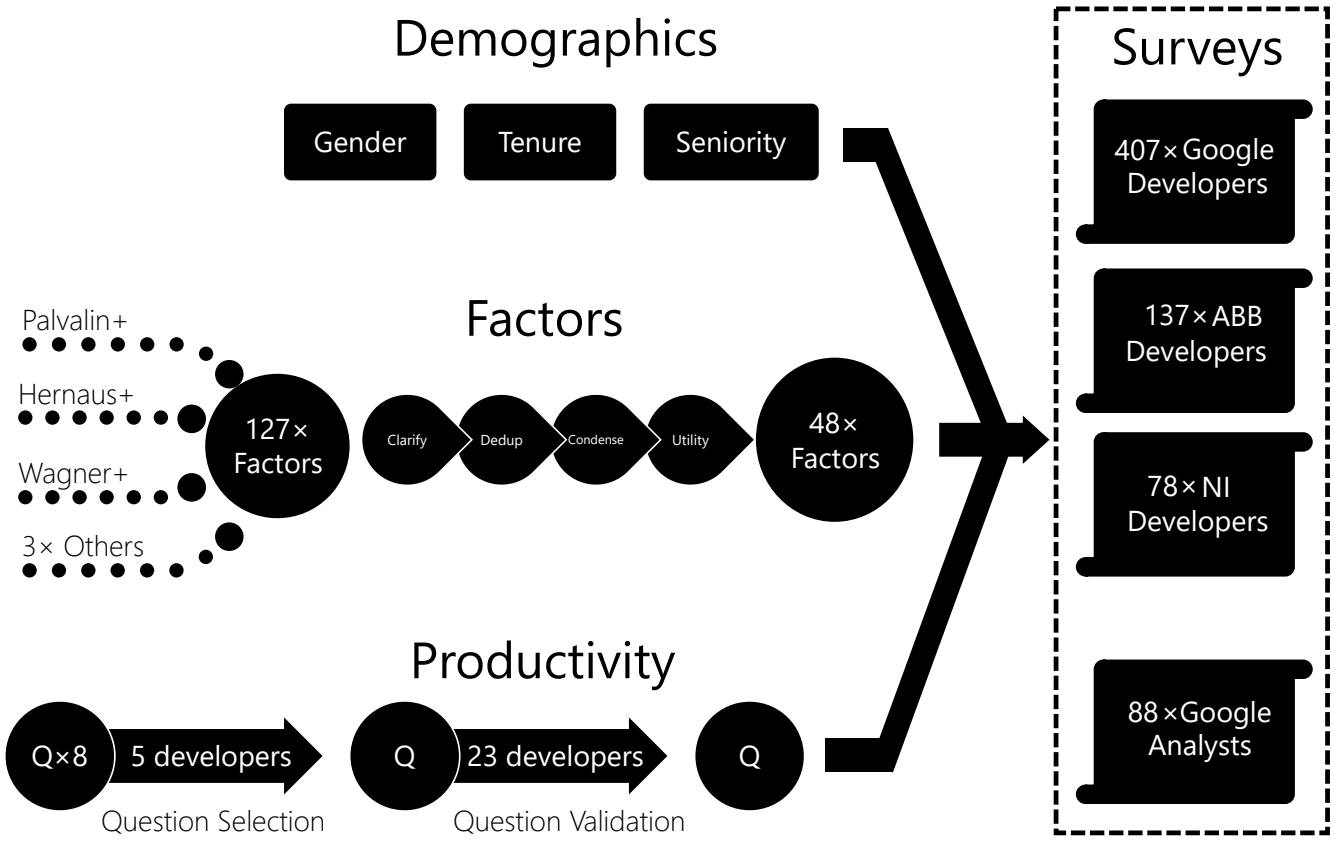

Fig. 1: Methodology used to create survey.

candidate questions, then narrowed them to this one by informally interviewing five developers at Google about their interpretation of the question (Figure 1, bottom left). For the question we included the qualifiers "high" and "regularly" in the question is a design choice, for three reasons. First, we wanted to establish a fixed point to which they could compare themselves. Second, we wanted that point to be a high benchmark, as a way of avoiding ceiling effects in responses. Third, we wanted to focus responses on two specific dimensions of productivity, namely intensity and frequency. Future researchers could obtain a more granular productivity measurement by disaggregating intensity and frequency in two separate questions.

We field tested the question by asking three managers at Google to send this question to their software development teams, along with the question "What did you consider when answering this productivity question?" Twenty-three developers responded (Figure 1, bottom center). We deemed the question acceptable for our purposes because the respondents' considerations when answering the questions were congruent with our expectations about the meaning of productivity, encompassing issues regarding workflow, work output, being in the zone or flow, happiness, goals achieved, coding efficiency, progress, and minimizing waste. Although we do not analyze them in this paper, the survey also included four additional, narrower measures of productivity, drawn from prior work [2], [4], [5].

To provide some objective data to contextualize our self-rated productivity measure, we chose two convenient objective measures of productivity, then correlated them with self-rated productivity within Google. The first objective measure is lines of code changed per unit time by a developer, a commonly used but fraught measure of software engineering productivity [6], [7]. The second is number of changelists merged into Google's main codebase per unit time by a developer, a measure essentially equivalent to Vasilescu and colleagues' measure of pull requests accepted per month [8]. For self-reported productivity, in this analysis we use responses to a similar survey at Google ($n = 3344$ responses); we could not use the data from our survey in this analysis because the original responses did not include participant identifiers that we could use to then link with objective productivity measures. This similar survey question asked, "How often do you feel you reach a high level of productivity at work?"; participants could answer Rarely or never, Sometimes, About half the time, Most of the time, and All or almost all the time. We then create a linear regression with self-rated productivity as an ordinal dependent variable (coded 1, 2, 3, 4, and 5, respectively).[2] As independent variables, we use each objective measure, log scaled since they are both positively skewed. For control variables, we included job code (e.g., software engineer, research engineer, etc.) as a categorical variable and seniority as a numeric level (e.g., level 3 is an entry level software engineer

2. Linear regression assumes that the distance between productivity ratings are equal. Given the question's wording, we believe this assumption is reasonable. Ordered logistic regression does not require this assumption. Applying that technique here yields robust results: the same coefficients are significant in linear and ordered models.

Model	Factor	Estimate	Sig.	R^2
1	log(lines_changed + 1)	0.045	***	0.017
	level	-0.051	**	
2	log(changelists_created + 1)	0.112	***	0.024
	level	-0.050	**	
3	log(lines_changed + 1)	-0.015	n.s.	0.024
	log(changelists_created + 1)	0.132	***	
	level	-0.051	**	

Fig. 2: Models predicting a subjective productivity measure from two objective measures. n.s. indicates a non-significant factor with $p > .05$, ** indicates $p < .01$, *** indicates $p < .001$. Full models are described in the Supplementary Material.

at Google). Job code was statistically significant for two job roles in each linear model. We ran three models; two with each objective productivity measure in isolation, and one with both objective measures.

The results of this contextualization are shown in Figure 2. Each model shows that level is statistically significant, with a negative estimate, which we interpret to mean more senior developers tend to rate themselves slightly less productive. This provides a strong rationale for us to control for seniority (Section 3.7). The first two models indicate significant positive correlations between each objective measure of productivity and subjective productivity; this suggests that the more lines of code modified and changelists submitted, the more productive an engineer self-rates. The final unified model, as well as the estimates on the first two models, suggests that number of changelists submitted is a stronger predictor of self-rated productivity than lines of code submitted. However, on each model, notice that the R^2, which represents the percent of variance explained, is quite low at less than 3% for each model. Overall, these results suggest that the number of lines of code and changelists submitted are part of engineers' self-estimate of their productivity, but only a small part.

3.2 Productivity Factors

Next on our survey, we asked participants about factors that prior literature suggests correlate with productivity. We drew these questions from four sources, as illustrated at the center left of Figure 1. We chose these sources because, to our knowledge, they represent the most comprehensive reviews of individual productivity factors in the software engineering and knowledge work research literature.

The first source we used was Pavalin and colleagues' instrument, which the authors derived from a literature review of performance measurements for knowledge workers [4]. The standardized survey instrument, called SmartWoW, was fielded at four organizations, and encompass aspects of the physical workspace, the virtual workspace, the social workspace, personal work practices, and well-being at work. We modified some of the questions to better reflect contemporary terminology in software engineering and more conventional American English. For instance, SmartWoW asks:

> I often telework for carrying out tasks that require uninterrupted concentration.

Which we rephrased as:

> I often work remotely for carrying out tasks that require uninterrupted concentration.

SmartWoW included 38 candidate questions for our survey.

The second source we used was Hernaus and Mikulić's literature review of work characteristics' effects on knowledge worker productivity [9]. Their survey, validated for reliability, is itself an amalgamation of prior research on productivity, including the Work Design Questionnaire [10], the Job Diagnostic Survey [11], Campion and colleagues' measurements of group cooperation [12], and Hernaus' own measure of the "nature of the task" [13]. We modified questions directly from these underlying instruments for brevity and consistency. In a similar vein, we considered using Campion and colleagues' instrument directly, but it focused on characteristics of work groups with little considerations for individuals' productivity.

The third source was Wagner and Ruhe's structured review of productivity factors in software development [14]. Unlike the other sources, this paper was not rigorously peer reviewed and does not contain original empirical studies, but to our knowledge it is nonetheless the most comprehensive review of studies of productivity in software engineering. Wagner and Ruhe's factors are broken down into technical factors and soft factors, and further broken down into environmental; team and corporate culture; project, product, and development environment; capability; and experience factors.

A fourth source we used was Meyer and colleagues' survey of Microsoft developers, which listed the top five reasons that developers have productive workdays, including goal setting, meetings, and interruption [15].

We also opportunistically included three additional factors that we felt were not adequately captured in the prior work, that were of particular interest to Google. One was a factor from the Knowledge Work Productivity Assessment [16], an unpublished precursor to SmartWoW, which we adapted as:

> The information supplied to me (bug reports, user stories, etc.) is accurate

The second was a factor directly from the Work Design Questionnaire, adapted as

> I receive useful feedback about my job performance

We created the third factor, which was of particular interest to ABB:

> I require direct access to specific hardware to test my software.

In total, we had 127 candidate factors. To reduce these to a manageable number of questions that respondents could answer without significant fatigue [17], we used the following criteria for inclusion, as illustrated at the center of Figure 1:

1) Eliminate duplicates. For instance, both SmartWoW [4] and Meyer and colleagues [15] discuss goal setting as an important productivity factor.
2) Condense similar factors. For instance, Hernaus and Mikulić describe multiple aspects of communication between work groups that improve productivity, but we condensed these down to a single factor [9].
3) Favor factors with clear utility. For instance, SmartWoW [4] includes the factor

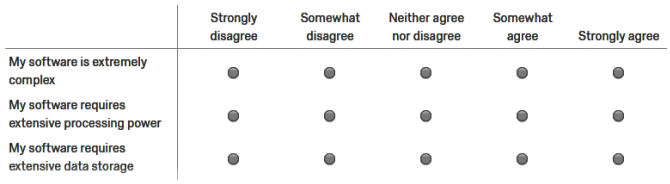

Fig. 3: Example factor questions from the survey.

> Workers have [the] opportunity to see each other's calendar
>
> At Google, this is universally true and unlikely to change, so the factor has little utility.

We applied these criteria collaboratively and iteratively. First, we printed a large physical poster of all candidate questions. Then, we put that poster on a wall at Google, near the Google authors' shared office space. Next, each Google author independently reviewed and revised the questions on the poster based on the criteria above. The poster remained on the wall for several weeks as we periodically added and revised feedback. Finally, the first author applied that feedback to create the final question set.

In the end, our survey included 48 factors in the form of statements (Figure 4, left column), which respondents marked their level of agreement with on a five point scale, from Strongly disagree to Strongly agree. These factors can be categorized into blocks about practices, focus, experience, job, work, capabilities, people, project, software, and context. An example question block is shown in Figure 3. We also asked one open-ended question about factors that respondents felt that we had missed. A full blank survey is available in the Supplementary Material.

3.3 Demographics

We collected and asked about several demographic factors that we wanted to control for, as illustrated at the top of Figure 1:

- Gender
- Tenure
- Seniority

We asked about gender because prior work suggests that gender is related to software engineering-relevant productivity factors, such as success in debugging [18]. Thus, the survey asked an optional question about gender (Male, Female, Decline to State, Custom). Respondents who did not answer the question were recoded to "Decline to State" (Google $n = 26$ [6%], ABB $n = 4$ [3%], National Instruments $n = 5$ [6%]). We treated this data as categorical.

For tenure, we collected years at the company at Google directly from human resources data. This was not feasible at ABB and National Instruments, so we added an optional question about it to their survey. At ABB, we recoded missing data ($n = 4$ [3%]) as 12 years of experience, the mean of the reported data. At National Instruments, we use 9 years ($n = 1$ [1%]), for the same reason. More sophisticated data imputation techniques are possible [19], such as by using multiple imputation to predict missing values based on present values; for instance, missing seniority values can likely be imputed somewhat accurately based on tenure and gender. Instead of using such techniques, we chose to use the simple statistical mean for imputation because demographic factors were not primary variables of interest, but instead simply covariates to control for. We treated this data as numerical.

For seniority, at Google the survey asked participants to optionally report their numeric job level. We recoded missing responses ($n = 26$ [6%]) as the most common level of the reported data. For ABB, participants could optionally mark Junior or Senior Software Engineer/Developer, although many participants marked "other" job titles. For these, we recoded responses as Senior if the titles contained the words senior, lead, manager, architect, research, principal, or scientist; we recoded them as Junior otherwise. Non-responses ($n = 4$ [3%]) we recoded as the most common response, Senior. For National Instruments, the options were Entry, Staff, Senior, Principal Architect/Engineer, Chief Architect/Chief Engineer, Distinguished Engineer, Fellow, or other. The single "other" response reported being an intern, which we recoded as "Entry." Non-responses ($n = 3$ [4%]) we recoded as the most common response, Senior. We treated seniority data from each company as numerical.

3.4 Non-Developer Comparison Survey

In our third research question, we wish to know what predicts developer self-rated productivity in particular. For example, we expect that interruption affects productivity, but we would also expect that interruption affects the productivity of any knowledge worker, so one would naturally ask whether interruption is particularly influential for developer productivity.

To answer this research question, we aimed to select a population comparable to software developers. We began by trying to find knowledge workers by inspecting common job titles at Google. While a number of candidate titles seemed to reasonably indicate knowledge work, the most common and, in our judgment, most reliable indicator of non-developer knowledge worker was whether the employee had "analyst" in their title. We chose to compare Google analysts to Google developers, instead of comparing Google analysts to developers across all three companies. Our reasoning is that doing so will control for any company-specific effects (e.g., perhaps Google employees are statistically more or less likely than employees at other companies to be sensitive to interruption).

We next adapted our survey to analysts. This included eliminating questions that were clearly software development-specific, such as "My software's requirements change frequently". Other questions we adapted specifically for analysts; for instance, we adapted "I use the best tools and practices to develop my software" to "I use the best tools and practices to do my job".

Self-rated productivity was measured the same way for analysts as for developers. We measured gender, tenure, and seniority the same way as we did for developers at Google. We piloted the analyst version of the survey with a convenience sample of five analysts, who reported the overall survey was clear, but also made some minor suggestions. We adjusted the survey accordingly before deploying it.

3.5 Attention Question

In an attempt to eliminate survey responses that were not given due consideration, we included an attention item [20] about two-thirds of the way through the survey. The attention question said "Respond with 'Somewhat disagree' to this item." Surveys that did not include a response of 'Somewhat disagree' to this attention item were discarded.

3.6 Response Rates

We used human resources data from Google to select 1000 random full-time employees with software developer job codes. We received 436 responses, a 44% response rate, which is very high for software engineering surveys [21]. 407 valid surveys remained after removing those that answered the attention question incorrectly ($n = 29$ [7%]).

For knowledge workers, we selected 200 random full-time employees at Google with "analyst" in their job titles. We chose not to survey more analysts because we judged it unethical to survey more analysts, when the survey results are primarily of benefit to software developers. A total of 94 analysts responded, a 47% response rate. 88 valid surveys remained after removing those that answered the attention question incorrectly ($n = 6$ [6%]).

We sent the survey to about 2200 randomly selected software developers at ABB, and received 176 completed surveys. This 8% response rate is on the low end of typical software engineering surveys [21]. 137 valid surveys remained after removing those that answered the attention item incorrectly ($n = 39$ [22%]).

We sent the survey to a mailing list containing about 350 software developers at National Instruments, and received 91 completed surveys (26% response rate). 78 valid surveys remained after removing those that answered the attention question incorrectly ($n = 13$ [14%]).

3.7 Analysis

In our analysis, we run linear individual multiple regression models for each factor at each company, with the factor as an independent variable (e.g. "My project deadlines are tight") and self-rated productivity as the dependent variable. We chose to run separate models at each company for privacy, so that each company's raw data was not shared with any other company. To reduce the influence of covariates, we included available demographic variables in each regression model. To interpret the results, we focus on three aspects of the productivity factor coefficient:

- **Estimate.** This indicates the effect size of each factor while holding the demographics constant; the further this number is from zero, the greater the effect.
- **Standard Error.** This represents the variability of the estimate; lower numbers represent less variability.
- **Significance.** We also analyze statistical significance, using $p < .05$ as a threshold. Because we are running 48 statistical tests per company and thus are likely to discover a handful of relationships simply by chance, we correct such p-values on a per-company basis using the Benjamini-Hochberg method, a technique designed to correct for false discovery [22].

In our interpretation of the results, we focus more on effect size (that is, the estimate) and less on the statistical significance, because statistical significance can be obtained from sufficiently large data sets, even when practical significance is low. As we shall see in the results, statistically significant results are obtained most frequently at Google, where our response rate was high, and least frequently at National Instruments, where our response rate was lower. Thus we interpret this difference as a largely a matter of statistical power. We encourage the reader to place more confidence in results with statistical significance.

To provide context, we also analyze to what extent demographic factors correlate with self-reported productivity. We do this by running multiple linear regression, with the demographic variables at each company as independent variables, and self-reported productivity as the dependent variable. We then analyze the overall predictive value of the resulting model, as well as how each independent variable contributes to the overall model.

3.8 A Word on Causality

Our methodology measures correlations between productivity factors and self-reported productivity, but fundamentally we are actually interested in measuring the degree to which each factor causes productivity changes. So to what extent is it valid to assume that there actually exists causal links between the factors and productivity?

Validity depends largely on the strength of the evidence for causality in the prior work. The strength of this evidence varies from factor to factor. As one example, Guzzo and colleagues' meta-analysis of 26 articles about appraisal and feedback provides excellent evidence that feedback does indeed cause productivity increases in the workplace [23]. However, establishing the strength of the evidence for every factor we surveyed would require an extensive meta-review, which is well beyond the scope of the present paper.

In sum, although our survey itself cannot establish causality, by relying on prior work, we can have some confidence that these factors cause increases or decreases in productivity, but the reader should interpret our results with some caution.

4 Results

To describe the results of deploying our study, we begin by describing the correlation between each productivity factor and self-rated productivity when controlling for demographics. We use these results to answer each research question in turn, followed by a discussion of the results of each question. We then discuss the relationship between the demographics and self-rated productivity. Finally, we discuss implications and threats.

4.1 Productivity Factors

Figure 4 shows the results of our analysis, which we outlined in Section 3.7. The first column indicates the factor, in the form of the statement shown to software developers on the survey, followed by factor labels (F1, F2, and so on) that we assigned after we completed data analysis. Missing data for certain factors means that the factor was developer-specific, and thus was not presented to analysts (for example, F10).

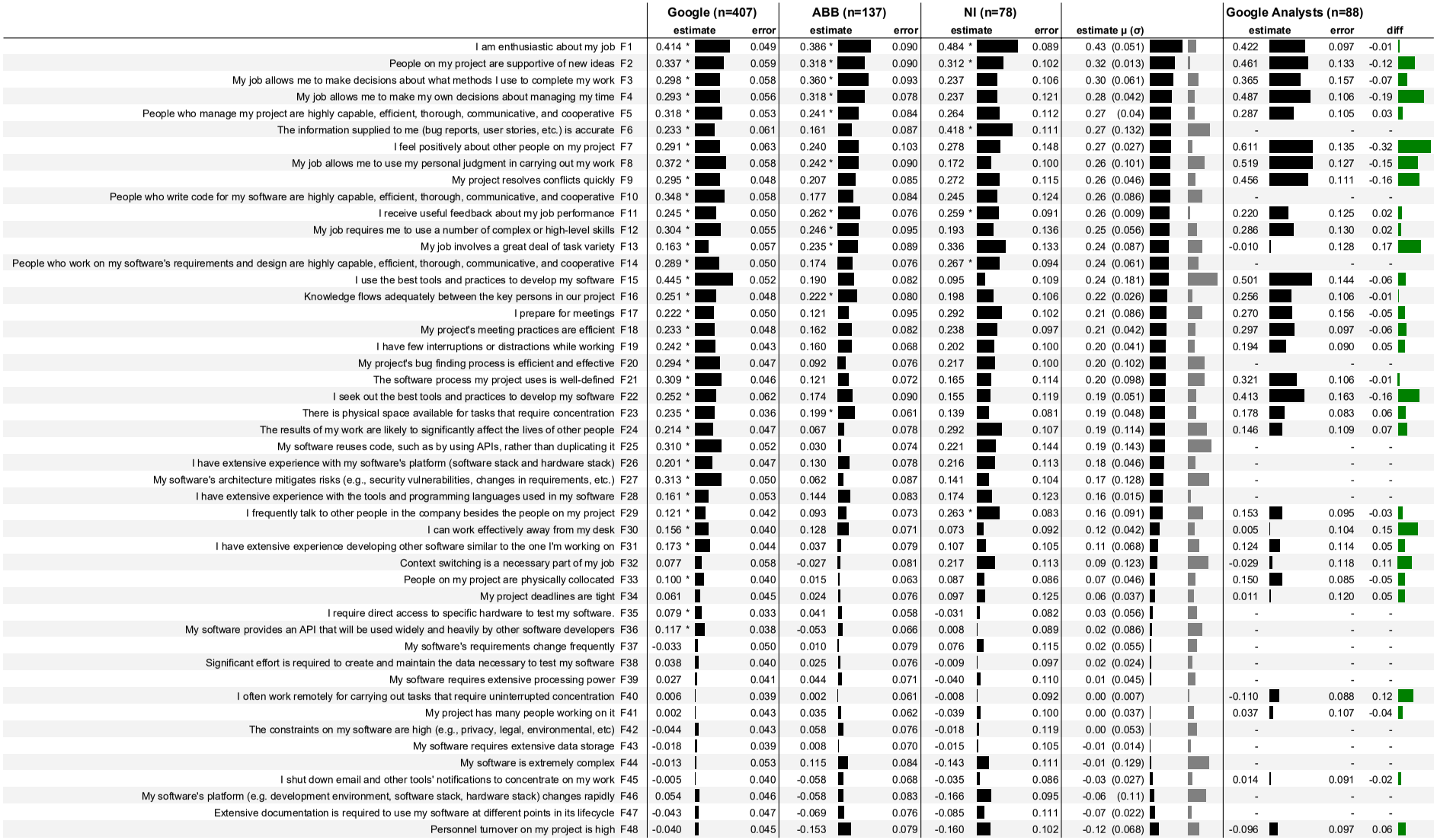

Fig. 4: 48 factors' correlation with developers' and analysts' self-rated productivity at three companies.

The other major columns display data from each of the three labeled companies, plus data about Google's analysts. Within each major column, there are two minor columns.

The minor column labeled **estimate** indicates regression coefficient, which quantifies the strength of the association between that factor and self-rated productivity, where the larger the number, the stronger the association. For example, the first row for Google gives the estimate as 0.414. Concretely, this means that for every one point increase in agreement with the statement about job enthusiasm (F1), the model predicts respondents' productivity rating will increase by 0.414 points, controlling for demographic variables. Estimates can likewise be negative. As an example, for all 3 companies, the more personnel turnover on a team (F48), the lower the self-rated productivity. Adjacent to each estimate is a single bar that reflects that magnitude of the estimate.

Note that estimate does *not* mean higher factor ratings, but instead means a higher correlation between a factor and self-rated productivity. For instance, National Instruments's estimate for job enthusiasm (F1) is higher than that of the other two companies. This does *not* mean that developers are the most enthusiastic at National Instruments; what it does mean is that developers' job enthusiasm was a stronger predictor of self-rated productivity at National Instruments than at the other two companies. We do not report ratings directly because it was a condition of corporate participation. In particular, ratings could be interpreted wrongly without full context; for instance, if we reported that one company's developers had a lower job enthusiasm rating than another company's developers, the reader may get the impression that the latter company is not a good company to work at.

The minor column labeled **error** indicates the standard error of the model for each factor. Lower numbers are better. Intuitively, lower numbers indicate that as the factor changes, the model will more reliably predict self-rated productivity. Overall error rates are fairly consistent across factors, especially for Google, which had a larger number of respondents.

Estimate and standard error values with an asterisk (*) indicate that the factor was statistically significant in its model. For instance, job enthusiasm (F1) was statistically significant across all three companies, but meeting preparation (F17) was only significant in Google.

The third major column indicates the mean ($\boldsymbol{\mu}$) estimate across all three companies, with the standard deviation in parentheses ($\boldsymbol{\sigma}$). The first bar visualizes the magnitude of the mean estimate, and the second bar visualizes the magnitude of the standard deviation. For example, the average estimate was 0.43 for job enthusiasm (F1), with a standard deviation of 0.051. The table is sorted by the mean estimate.

The last column indicates the magnitude of the difference (**diff**) between estimates at Google for software developers versus analysts. Positive values indicate developers' estimates being higher, negative values indicate analysts' estimates being higher. For example, analysts' estimates tended to be slightly lower than developers' for enthusiasm (F1).

4.2 RQ1: What factors most strongly predict developers' self-rated productivity?

The strongest predictors of self-rated productivity are the statements with the highest absolute mean estimate; the weakest factors of self-rated productivity are those with the lowest absolute mean estimate. In other words, the factors at the top of the Figure 4 are the strongest predictors, and at the bottom, the weakest. To determine which of the factors we can have the most confidence in, we identify the results that are statistically significant across all three companies:

- Job enthusiasm (F1)
- Peer support for new ideas (F2)
- Useful feedback about job performance (F11)

Discussion. A notable outcome of the ranking is that the top 10 productivity factors are non-technical. This is somewhat surprising, given that most software engineering research tends to focus on technical aspects of software engineering, in our estimation. Thus, a vigorous refocusing on human factors may yield substantially more industry impact for the software engineering research community. For instance, answering the following questions may be especially fruitful:

- What makes software developers enthusiastic about their job? What accounts for differences in levels of enthusiasm between developers? What interventions can increase enthusiasm? This work can extend existing work on developer happiness [24] and motivation [25].
- What kinds of new ideas are commonly expressed in software development practice? What actions influence developers' feelings of support for those ideas? What interventions can increase support for new ideas, while maintaining current commitments?
- What kinds of job feedback do software engineers receive, and what makes it useful? What kinds of feedback is not useful? What interventions can increase the regularity and usefulness of feedback?

Another notable feature is the ranking of factors from the COCOMO II line of research. COCOMO II factors, derived from empirical studies of industry software projects and validated through quantitative analysis of 83 projects [26], were originally designed to help estimate the cost of software projects. Examples of COCOMO II productivity factors include the volatility of the underlying platform and the complexity of the product. Interestingly, the COCOMO II factors in our survey (F5, F10, F14, F16, F24, F26, F28, F32, F33, F34, F36, F38, F39, F43, F44, F46, F47, F48) tend to be lower ranked, suggesting they are less predictive of productivity than the other factors. Of the top 50% of most predictive factors (F1–F24), only 5 are related to COCOMO II; in the bottom 50%, 14 are. We see two different interpretations of this result. One is that COCOMO II is missing several important productivity factors, and that future iterations of COCOMO II may be able to make more accurate predictions if they incorporated the more predictive productivity factors we studied, such as an organization's support for work method autonomy. Another interpretation is that COCOMO II is fit for its current purpose – capturing productivity at the project level [6], [27], [28], [29], [30], [31] – but it is less suitable for capturing productivity at the individual developer level. This interpretation underscores the importance, and the novelty, of our results.

Similarly, all of COCOMO II's platform factors were relatively small and not statistically significant predictors of productivity across all three companies studied, such as:

- My software requires extensive processing power (F39)
- My software requires extensive data storage (F43)
- My software's platform (e.g. development environment, software stack, hardware stack) changes rapidly (F46)

One explanation is that, in the 20 years since COCOMO II was created and validated, platforms have begun to make less of a difference in terms of productivity. Standardized operating systems now arguably shield developers from productivity losses due to hardware changes (e.g., Android for mobile development). Likewise, cloud computing platforms may shield developers from productivity losses due to scaling up processing and data storage needs. This is not to say that modern operating system frameworks and cloud computing platforms are easy to use, but instead that the productivity gap between processing a little data and processing a lot of data may have closed since COCOMO II.

4.3 RQ2: How do these factors differ across companies?

To answer this question, we can look to the standard deviation of estimates across all three companies. The three factors with the lowest variance, that is, were the most consistent across companies were:

1) Use of remote work to concentrate (F40)
2) Useful feedback about job performance (F4)
3) Peer support for new ideas (F2)

We posit that the stability of these factors makes them good candidates for generalizability; companies beyond the ones we studied are likely to get similar results as to ours, for these factors.

The three factors with the highest variance, that is, varied the most between companies were:

1) Use of best tools and practices (F15)
2) Code reuse (F25)
3) Accuracy of incoming information (F6)

Discussion. The three factors that varied the least between companies (F40, F4, and F2) have a common feature that they are social and environmental, rather than technical. Perhaps this suggests that developers, regardless of where they work, are equally affected by remote working, job feedback, and peer support for new ideas. Interventions that change these three factors may have a higher impact than other interventions, since these factors seem so consistent.

And why might F15, F25, and F6 vary so much between companies? We provide a potential explanation for each, based on what we know about these companies.

Using the best tools and practices (F15) was most strongly related to self-rated productivity at Google, but weakly, non-significantly related in National Instruments. One potential explanation is that Google's codebase may be substantially larger than National Instruments'. Consequently, using best practices and tools to effectively navigate and understand a larger codebase is essential to productivity; at National Instruments, productivity is less sensitive to tool use due to the smaller, more understandable nature of the codebase.

Reusing existing code (F25) was strongly related to self-rated productivity at Google, but weakly, non-significantly related at ABB. One potential explanation is that code reuse is easier to achieve in Google than in ABB. Google uses a monolithic codebase, where all developers can browse nearly every line of code in the company, so reusing code comes with little effort. In contrast, ABB uses multiple repositories, so reusing code across the company requires developers to gain access to other repositories before reusing code from them. Thus, in ABB, productivity gains (achieved through code reuse) may be negated by productivity losses (in finding and accessing relevant code to reuse).

Accurate information (F6) was strongly related to self-rated productivity at National Instruments, but weakly, non-significantly related at ABB. One potential explanation was that developers at ABB are more insulated from the effect of inaccurate information than at National Instruments. Specifically, ABB has multiple levels of support teams dedicated to getting correct information from customers about bugs. If developers at ABB receive inaccurate information, their productivity may not be reduced because they can delegate correcting that information back to the support teams.

4.4 RQ3: What predicts developer self-rated productivity, in particular, compared to other knowledge workers?

To answer this question, we can look at the last column of Figure 4. Looking at a few of the correlations with the highest estimates, we can see that, compared to software developers, analysts' self-rated productivity was more strongly related to:

- Positive feelings about their teammates (F7)
- Time management autonomy (F4)

On the other hand, developers' self-rated productivity was more strongly related to, for example:

- Doing a variety of tasks as part of their work (F13)
- Working effectively away from their desks (F30)

Discussion. Overall, these results suggest that developers are similar to knowledge workers in some ways and different from them in others. For instance, the strongest predictor of productivity for developers was job enthusiasm, but that factor predicted productivity to nearly the same extent for analysts. We believe that our results on the similarities and differences between developers and other knowledge workers can be used by organizations either to choose productivity initiatives targeted specifically at developers or to choose initiatives targeted more broadly.

The unified development toolset at Google may explain why an increase in task variety corresponds to a larger self-rated productivity increase for developers than analysts. While task variety may decrease boredom and thus increase productivity in both groups, the unified toolset for developers at Google may mean that developers can use the same tools for different tasks. In contrast, analysts may have to use different toolsets for different tasks, inducing a heavier penalty for mental context switching.

The disruption caused by interruption may explain why why an increase in working effectively away from one's desk corresponds to a larger productivity increase for developers than analysts. Specifically, we posit that perhaps interruption is more harmful during programming than during analytical work, so being able to work effectively away from one's desk may be especially important for developers' productivity.

Parnin and Rugaber found that resuming tasks after interruption is a frequent and persistent problem for developers [32], caused in part by the need for better tools to assist with task resumption [33].

4.5 Other Productivity Factors

At the end of the survey, respondents were able to respond to write-in additional factors that they felt impacted productivity. We noticed that these responses largely provided the same or more specific descriptions of our 48 factors, so we closed-coded responses based on our factors, and created a handful of new factors when necessary. We provide an accounting of the new factors, as well as the more specific descriptions of our factors, in the Supplementary Material. Overall, researchers deploying and modifying our survey instrument may wish to add a new question about a projects' personnel mix. They may also wish to either clarify or provide more specific breakdowns of the questions for factors F15, F16, and F19.

4.6 Demographics

For Google and National Instruments, neither the overall demographic models nor any individual covariates were statistically significant predictors of self-rated productivity. For ABB, the demographic model was significant ($F = 3.406, df = (5, 131), p < .007$). Gender was a statistically significant factor ($p = .007$), with female respondents reporting 0.83 points higher self-rated productivity than males, and custom genders reporting 1.6 points higher self-rated productivity than males ($p = .03$). Tenure also showed significant results ($p = .04$), with self-rated productivity increasing .02 points for every additional year at the company. To our knowledge, the differences between ABB and the other two companies do not seem to explain why these demographic factors were significant predictors at ABB but not elsewhere.

4.7 Implications for Practice and Research

How should practitioners put our results into action? We provide a ranked list of most predictive productivity factors, which can be used to prioritize productivity initiatives. Such initiatives can be drawn from prior work. For example, to increase job enthusiasm, Markos and Sridevi suggest helping workers grow [34], such as by providing workshops to technology and interpersonal skills. Markos and Sridevi also suggest providing recognition for good work; for instance, ABB has experimented with providing public recognition for developers who adopt tools and practices for structured code navigation [35]. To increase support for new ideas, Brown and Duguid suggest informal and formal sharing of best practices [36]. At Google, one way knowledge is shared is through an initiative called "Testing on the Toilet", where engineers write short newsletters about testing or other technical knowledge, and those newsletters are posted in bathrooms around the company. To increase feedback quality about job performance, London and Smither suggest focusing on feedback that is non-threatening, behaviorally-focused, interpreted, and outcome-oriented [37]. At Google, one way that feedback is provided in this way is through *blameless postmortems*, where after a significant negative outcome occurs like a service outage, engineers collaboratively write a report that focuses on the behaviors that contributed to a root-cause, without blaming individual engineers.

For researchers, several areas of future work in this area would build on this work and advance the field of software developer productibity. First, a systematic literature review, which characterizes the strength and context of the evidence for each productivity factor discussed in this paper, would improve actionability of this work by helping establish causal links. In areas where those links are weak, actionability would likewise be improved by conducting a set of experiments that establish causality. Second, as we mention in Sections 4.5 and 4.6, future researchers may want to consider other factors that our participants suggested, and investigate how gender and other demographic factors interact with developer productivity. Third, the impact of productivity research in software engineering would be improved with a multi-dimensional toolbox of productivity metrics and instruments, validated through empirical study and triangulation. Fourth, if researchers can quantify the cost to change the factors that drive productivity, organizations can make even smarter investment choices.

4.8 Threats

Several threats to the validity of this study should be considered when interpreting its results.

4.8.1 Content Validity Threats

First, we reported on only one dimension of productivity, self-rated productivity. Other dimensions of productivity exist, including objective measures, such as number of lines of code written per day per engineer, a measure used by companies like Facebook [38]. As we argue in Section 3.1, all productivity metrics have drawbacks, including self-rated productivity. For instance, developers may be insufficiently self-aware of their own productivity or may artificially increase their self-rating due to social desirability bias [39]. Despite this limitation, Zelenski and colleagues draw from prior work to argue for the validity of a single item, self-report measure of productivity [40], such as the one we use in our paper.

Second, we measured self-reported productivity through a single question, which likely does not capture the full range of developers' productivity experiences. For instance, the question's wording focuses on frequency and intensity, but omits other aspects of productivity like quality. The survey also did not ask respondents to bound their answers by a particular time period, so some participants may have provided answers that reflect their experience over the last week whereas others reflect their experience over the last year. In retrospect, the survey should have specified a fixed time window.

Third, because we were limited in the number of questions we could ask and we relied on only the factors studied in prior work, our 48 chosen factors may not capture all aspects of behavior that influence productivity. Likewise, our chosen factors may be too broad in some cases. For example, in retrospect, the factor that asks about best "tools and practices" (F14) would probably be more actionable if it disaggregated tools from practices.

4.8.2 Internal Validity Threats

Fourth, as we mentioned in Section 3.8, we rely on prior work to establish a causal link between factors and productivity,

but the strength of the evidence for causality varies. For some factors, it may be that the factor and self-rated productivity are only linked through some third factor, or that the causal relationship is reversed. For instance, it's plausible that the top productivity factor, increased job enthusiasm (F1), could actually be caused by increased productivity.

4.8.3 External Validity Threats

Fifth, although we surveyed three fairly diverse companies, generalizability to other types of companies, to other organizations, and to other types of knowledge workers, is limited. Similarly, in this paper we selected analysts to represent non-developer knowledge workers, but this selection excludes several types of knowledge workers, such as physicians, architects, and lawyers. Another threat to validity is non-response bias; the people who responded to the survey were self-selected.

Sixth, we analyzed each productivity factor in isolation, but multiple factors may co-vary. This is not a problem so much with the analysis, but with its actionability; if factors are co-dependent, adjusting one may adversely affect another.

4.8.4 Construct Validity Threats

Seventh, in designing this survey we were concerned with respondents' ability to guess our analysis methodology and not answer truthfully as a result. We attempted to mitigate this by separating the productivity question from the productivity factors, but respondents nonetheless may have been able to infer our analysis methodology.

Finally, we reworded some questions to adapt the survey to analysts, which may have altered their meaning in unintended ways. Consequently, differences between developers and analysts may be unduly influenced by question differences, rather than job differences.

5 Related Work

Many researchers have studied individual factors of software developer productivity. One example is Moser and Nierstrasz' field study of 36 software projects, which examined whether object-oriented technology can improve software developer productivity [41]. Another example is DeMarco and Lister's study of 166 programmers in 35 different organizations performing a one-day programming exercise, which found that workplace and organization correlated with productivity [42]. A third example is Kersten and Murphy's lab experiment with 16 developers, which showed that developers who used a task-focusing tool were significantly more productive than those that did not use the tool [43]. Beyond these examples, Wagner and Ruhe's systematic review provides a good overview of the individual productivity factors that correlate with productivity [14]. Likewise, Meyer and colleagues provide a more recent overview of productivity factors [3]. In general, our work builds on these factor-by-factor productivity studies by broadly investigating a wide variety of factors.

As described in Petersen's systematic mapping literature review, seven software engineering papers have quantified the factors that predict software development productivity [44]. Each paper uses a quantitative method, typically a regression as we use in this paper, to predict productivity in several software projects. The most common factors relate to the size of the project and 6 of 7 explicitly build on COCOMO II productivity drivers [6], [27], [28], [29], [30], [31]. The most complex predictive model in Petersen's study is based on 16 factors [6]. Our paper differs in two main respects. The first is that we evaluate a larger number (48) and a wider variety of productivity factors than prior work, drawing heavily on industrial and organizational psychology. The second is unit of analysis; while these prior studies examined what predicts *project* productivity, we instead examine what predicts *people's* productivity.

Beyond software engineering, prior work has compared the factors that predict productivity in *other* kinds of workers, particularly in industrial/organizational psychology. While such research has investigated productivity at an organization-wide scale [45] and for physical work like manufacturing [46], the most relevant subfield is productivity of knowledge workers, that is, people who primarily work with knowledge and information, typically using a computer [47]. Two major pieces of work have compared factors for knowledge workers. The first is Palvalin and colleagues' investigation into 38 factors that had been shown to correlate with productivity in prior work; these factors cover the physical workplace, virtual workplace, social workplace, personal work practices, and well-being at work [4]. The second is Hernaus and Mikulić's survey of 512 knowledge workers, which investigated 14 factors in three categories [9]. We built on both of these studies when designing our survey (Section 3.2).

At the same time, studies that compare productivity factors for knowledge workers are insufficient for studying software developers for two main reasons. First, the degree to which these general results translate to software developers specifically is unclear. Second, such works tend to abstract away software-specific factors, such as software reuse and codebase complexity [48]. Thus, an important gap in the literature is a broad understanding of the factors that predict productivity for software developers. Filling this gap has practical importance; the authors of the present paper are part of three research groups across three different companies tasked specifically with improving developer productivity. Filling this gap helps our groups, and the companies we are part of, decide how to invest in developer productivity improvements.

6 Conclusion

Many factors influence developers' productivity, yet organizations have limited resources to invest in improving productivity. We designed and deployed a survey at three companies to rank and compare productivity factors. To focus their effort, developers, management, and executives can use our productivity factor ranking to prioritize what is otherwise a wide array of investment options.[3] In short, prior research provides many ways that organizations can improve software developer productivity, and our research provides a way to prioritize them.

3. Answers to the exercise from the introduction: 1a (see F24 vs. F45), 2b (see F44 vs. F3, F4, and F8), and 3a, unless your company is more like National Instruments than Google or ABB (see F15 vs. F19).

7 Acknowledgements

Thanks to our survey respondents, as well as Adam Meade, Rob Siemborski, Will Snipes, Mark Wilson, Thomas Zimmermann, and our anonymous reviewers.

References

[1] R. S. Nickerson, "Confirmation bias: A ubiquitous phenomenon in many guises." *Review of general psychology*, vol. 2, no. 2, p. 175, 1998.

[2] Y. W. Ramírez and D. A. Nembhard, "Measuring knowledge worker productivity: A taxonomy," *Journal of Intellectual Capital*, vol. 5, no. 4, pp. 602–628, 2004.

[3] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz, "The work life of developers: Activities, switches and perceived productivity," *IEEE Transactions on Software Engineering*, 2017.

[4] M. Palvalin, M. Vuolle, A. Jääskeläinen, H. Laihonen, and A. Lönnqvist, "Smartwow–constructing a tool for knowledge work performance analysis," *International Journal of Productivity and Performance Management*, vol. 64, no. 4, pp. 479–498, 2015.

[5] C. H. C. Duarte, "Productivity paradoxes revisited," *Empirical Software Engineering*, pp. 1–30, 2016.

[6] K. D. Maxwell, L. Van Wassenhove, and S. Dutta, "Software development productivity of european space, military, and industrial applications," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 706–718, 1996.

[7] J. D. Blackburn, G. D. Scudder, and L. N. Van Wassenhove, "Improving speed and productivity of software development: a global survey of software developers," *IEEE transactions on software engineering*, vol. 22, no. 12, pp. 875–885, 1996.

[8] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in github," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015, pp. 805–816.

[9] T. Hernaus and J. Mikulić, "Work characteristics and work performance of knowledge workers," *EuroMed Journal of Business*, vol. 9, no. 3, pp. 268–292, 2014.

[10] F. P. Morgeson and S. E. Humphrey, "The work design questionnaire (wdq): developing and validating a comprehensive measure for assessing job design and the nature of work." *Journal of applied psychology*, vol. 91, no. 6, p. 1321, 2006.

[11] J. R. Idaszak and F. Drasgow, "A revision of the job diagnostic survey: Elimination of a measurement artifact." *Journal of Applied Psychology*, vol. 72, no. 1, p. 69, 1987.

[12] M. A. Campion, G. J. Medsker, and A. C. Higgs, "Relations between work group characteristics and effectiveness: Implications for designing effective work groups," *Personnel psychology*, vol. 46, no. 4, pp. 823–847, 1993.

[13] T. Hernaus, "Integrating macro-and micro-organizational variables through multilevel approach," *Unpublished doctoral thesis). Zagreb: University of Zagreb*, 2010.

[14] S. Wagner and M. Ruhe, "A systematic review of productivity factors in software development," in *Proceedings of 2nd International Workshop on Software Productivity Analysis and Cost Estimation*, 2008.

[15] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann, "Software developers' perceptions of productivity," in *Proceedings of the International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 19–29.

[16] R. Antikainen and A. Lönnqvist, "Knowledge work productivity assessment," Tampere University of Technology, Tech. Rep., 2006.

[17] M. Galesic and M. Bosnjak, "Effects of questionnaire length on participation and indicators of response quality in a web survey," *Public opinion quarterly*, vol. 73, no. 2, pp. 349–360, 2009.

[18] L. Beckwith, C. Kissinger, M. Burnett, S. Wiedenbeck, J. Lawrance, A. Blackwell, and C. Cook, "Tinkering and gender in end-user programmers' debugging," in *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM, 2006, pp. 231–240.

[19] D. B. Rubin, *Multiple imputation for nonresponse in surveys*. John Wiley & Sons, 2004, vol. 81.

[20] A. W. Meade and S. B. Craig, "Identifying careless responses in survey data." *Psychological methods*, vol. 17, no. 3, p. 437, 2012.

[21] E. Smith, R. Loftin, E. Murphy-Hill, C. Bird, and T. Zimmermann, "Improving developer participation rates in surveys," in *Proceedings of Cooperative and Human Aspects on Software Engineering*, 2013.

[22] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: a practical and powerful approach to multiple testing," *Journal of the royal statistical society. Series B (Methodological)*, pp. 289–300, 1995.

[23] R. A. Guzzo, R. D. Jette, and R. A. Katzell, "The effects of psychologically based intervention programs on worker productivity: A meta-analysis," *Personnel psychology*, vol. 38, no. 2, pp. 275–291, 1985.

[24] D. Graziotin, X. Wang, and P. Abrahamsson, "Happy software developers solve problems better: psychological measurements in empirical software engineering," *PeerJ*, vol. 2, p. e289, 2014.

[25] J. Noll, M. A. Razzak, and S. Beecham, "Motivation and autonomy in global software development: an empirical study," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2017, pp. 394–399.

[26] B. Clark, S. Devnani-Chulani, and B. Boehm, "Calibrating the cocomo ii post-architecture model," in *Proceedings of the International Conference on Software Engineering*. IEEE, 1998, pp. 477–480.

[27] B. Kitchenham and E. Mendes, "Software productivity measurement using multiple size measures," *IEEE Transactions on Software Engineering*, vol. 30, no. 12, pp. 1023–1035, 2004.

[28] S. L. Pfleeger, "Model of software effort and productivity," *Information and Software Technology*, vol. 33, no. 3, pp. 224–231, 1991.

[29] G. Finnie and G. Wittig, "Effect of system and team size on 4gl software development productivity," *South African Computer Journal*, pp. 18–18, 1994.

[30] D. R. Jeffery, "A software development productivity model for mis environments," *Journal of Systems and Software*, vol. 7, no. 2, pp. 115–125, 1987.

[31] L. R. Foulds, M. Quaddus, and M. West, "Structural equation modelling of large-scale information system application development productivity: the hong kong experience," in *Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on*. IEEE, 2007, pp. 724–731.

[32] C. Parnin and S. Rugaber, "Resumption strategies for interrupted programming tasks," *Software Quality Journal*, vol. 19, no. 1, pp. 5–34, 2011.

[33] C. Parnin and R. DeLine, "Evaluating cues for resuming interrupted programming tasks," in *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 2010, pp. 93–102.

[34] S. Markos and M. S. Sridevi, "Employee engagement: The key to improving performance," *International Journal of Business and Management*, vol. 5, no. 12, pp. 89–96, 2010.

[35] W. Snipes, A. R. Nair, and E. Murphy-Hill, "Experiences gamifying developer adoption of practices and tools," in *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 105–114.

[36] J. S. Brown and P. Duguid, "Balancing act: How to capture knowledge without killing it." *Harvard business review*, vol. 78, no. 3, pp. 73–80, 1999.

[37] M. London and J. W. Smither, "Feedback orientation, feedback culture, and the longitudinal performance management process," *Human Resource Management Review*, vol. 12, no. 1, pp. 81–100, 2002.

[38] T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck, and M. Stumm, "Continuous deployment at facebook and oanda," in *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 21–30.

[39] R. J. Fisher, "Social desirability bias and the validity of indirect questioning," *Journal of consumer research*, vol. 20, no. 2, pp. 303–315, 1993.

[40] J. M. Zelenski, S. A. Murphy, and D. A. Jenkins, "The happy-productive worker thesis revisited," *Journal of Happiness Studies*, vol. 9, no. 4, pp. 521–537, 2008.

[41] S. Moser and O. Nierstrasz, "The effect of object-oriented frameworks on developer productivity," *Computer*, vol. 29, no. 9, pp. 45–51, 1996.

[42] T. DeMarco and T. Lister, "Programmer performance and the effects of the workplace," in *Proceedings of the International Conference on Software Engineering*. IEEE Computer Society Press, 1985, pp. 268–272.
[43] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 2006, pp. 1–11.
[44] K. Petersen, "Measuring and predicting software productivity: A systematic map and review," *Information and Software Technology*, vol. 53, no. 4, pp. 317–343, 2011.
[45] M. J. Melitz, "The impact of trade on intra-industry reallocations and aggregate industry productivity," *Econometrica*, vol. 71, no. 6, pp. 1695–1725, 2003.
[46] M. N. Baily, C. Hulten, D. Campbell, T. Bresnahan, and R. E. Caves, "Productivity dynamics in manufacturing plants," *Brookings papers on economic activity. Microeconomics*, vol. 1992, pp. 187–267, 1992.
[47] A. Kidd, "The marks are on the knowledge worker," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 1994, pp. 186–191.
[48] G. K. Gill and C. F. Kemerer, "Cyclomatic complexity density and software maintenance productivity," *IEEE transactions on software engineering*, vol. 17, no. 12, pp. 1284–1288, 1991.

Emerson Murphy-Hill received a PhD degree in computer science from Portland State University. He is currently a Research Scientist at Google. This work was performed while Emerson was a Visiting Scientist at Google.

Ciera Jaspan received a PhD degree in computer science from Carnegie Mellon University. She is currently a Software Engineer at Google.

Caitlin Sadowski received a PhD degree in computer science from the University of California, Santa Cruz. She is currently a Software Engineer at Google.

David Shepherd received a PhD degree in computer science from the University of Delaware. He is currently a Senior Principal Scientist at ABB.

Michael Phillips received a Bachelor of Science degree in computer engineering from Auburn University. He is currently the Head of Open Source and Developer Efficiency, Application and Embedded Software at National Instruments.

Collin Winter received a Bachelor of Science degree in computer science from Middle Tennessee State University. He is currently a Staff Software Engineer at Waymo.

Andrea Knight received a Masters of Human Computer Interaction from Carnegie Mellon University. She is currently a Staff UX Researcher at Google.

Edward Smith received Bachelor of Science degrees in computer science and psychology from the University of Maryland College Park. He is currently a Software Engineer at Bloomberg.

Matthew Jorde received a Masters degree in computer science from the University of Nebraska-Lincoln. He is currently a Software Engineer at Google.