

Enabling the Study of Software Development Behavior With Cross-Tool Logs

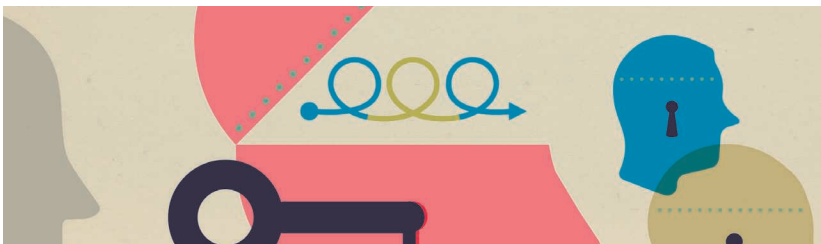
Ciera Jaspan, Matt Jorde, Carolyn Egelman, Collin Green, and Ben Holtz, Google

Edward Smith, Bloomberg

Maggie Hodges, Artech Information Systems, Google

Andrea Knight, Liz Kammer, Jill Dicker, Caitlin Sadowski, James Lin, Lan Cheng, Mark Canning, and Emerson Murphy-Hill, Google

// Capturing developers' behavior at scale can be challenging. In this article, we describe our experience creating a system that integrates log data from dozens of development tools at Google. //



HUMANS ARE ONE of the most expensive parts of software creation. While companies can always add more equipment, adding more

people comes with additional recruitment, training, and communication overhead. At Google, we have more than 30,000 developers, so if we can make even small improvements to individual developers' productivity, we can attain large overall

benefits for our products and their users. See “Actionable Insights” for a summary of what we learned building a system that combines data from multiple developer tools.

Motivation

At Google, the engineering productivity research team's goal is to help our engineers be more productive. To meet this goal, our main piece of technical infrastructure is a quantitative logs pipeline that we call *InSession*, which helps us understand the behavior of developers by answering questions such as the following:

- Does a developer's certification in a programming language improve productivity?
- Can we predict negative interpersonal interactions between code authors and reviewers?
- Do new version control systems help developers be more productive?

In this article, we describe *InSession*, which ingests logs from multiple developer tools to build a picture of a developer's behavior during their workday (see the supplementary material “Data Sources” on *IEEE Xplore* for additional details). Like integrated development environment (IDE) monitors, such as *Mylyn Monitor*¹ and *ABB's system*,² *InSession* captures behavioral data automatically, but it additionally captures data from tools outside of a developer's IDEs. Like multitool monitors, such as *HackyStat*³ and *PROM*,⁴ *InSession* combines behavioral data from multiple developer tools, but in contrast to these systems, *InSession* ingests existing cloud-based tool logs, meaning that developer behavior is captured on every workstation without the need

Digital Object Identifier 10.1109/MS.2020.3014573
Date of current version: 18 September 2020


This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see <https://creativecommons.org/licenses/by/4.0/deed.en>

to install custom sensor software for each individual tool.

Privacy Principles

To maintain the integrity of data and trust of employees, strong privacy principles are critical. In consultation with privacy experts, we identified the following core principles in the form of seven Dos and Don'ts:

1. **Do** ingest log data for only employees. InSession ingests data from employees' use of the tools and systems provided by Google for them to do their work.
2. **Do** focus on tools used for work purposes. Avoid collecting task-identifying metadata from tools with mixed purposes, such as email, but do collect such metadata for tools that are used exclusively for work purposes (e.g., code review).
3. **Don't** collect employee-generated content, with the exception of code repository paths, build rules, and similar artifacts that are available to the entire company.
4. **Do** encrypt stored data.
5. **Do** make stored data access auditable. We have an access logging system that tracks access to the raw files and their Structured Query Language (SQL)-table views.
6. **Don't** report data for individual employees without the individual's prior consent. Individuals must not be identifiable in reported aggregate data. Some may be tempted to include this data in performance evaluations. However, we do not do so, both on principle and because research suggests developers are opposed to such metrics.⁵
7. **Do** destroy data after a set retention period. We use a retention period of three years for individual data, which allows for



ACTIONABLE INSIGHTS

- Cross-tool log data offer an opportunity for organizations to understand the behaviors of software engineers.
- Google gained insights from building a logs-ingesting system called *InSession*, whose design is described in this article.
- We use InSession to show that an engineer certification process reduces the time engineers spend reviewing code.

year-over-year studies. Aggregated data are stored indefinitely.

Design and Implementation of InSession

Creating Events From Logs

We first discuss how InSession defines and classifies developer events.

cron jobs. Back-end events are useful for performing studies related to what developers do after launching a long-running action, e.g., do they switch to another task or continue with related work while waiting? Events can also be either *instantaneous* or *dura-tional*. An instantaneous event has an undefined end point, e.g., switching

To maintain the integrity of data and trust of employees, strong privacy principles are critical.

An *event* is a distinct usage of a tool or system by a developer or on a developer's behalf. Each log source has its own *importer*, which extracts information into our common event data format. Events come from both developer-specific tools, like bug tracking, version control, and editors, and from more general purpose tools, including Gmail and Calendar.

We distinguish between two types of event: *front end* and *back end*. A front-end event is one that a developer actively initiates, e.g., clicking a user interface button. A back-end event is one that occurs asynchronously on a developer's behalf, e.g.,

focus to a documentation page. A durational event has a set start and end time, e.g., running a build.

For most events, we also collect additional metadata about the event, which we call *artifacts*. These artifacts can be classified as *task-identifying* or *informational*. A task-identifying artifact is one that identifies a specific development task that can be used to group related events together, e.g., a development workspace label or identifier for a changelist (the proposed change to a codebase that generally undergoes review). An informational artifact provides contextual information about an event, e.g., the

path of a file viewed in a code search tool. Each event importer determines how artifacts from its log source should be classified. For example, the changelist identifiers for code review logs are always classified as task-identifying by their importers because it identifies the task of reviewing that changelist. The core artifacts we collect include changelist identifiers, development workspace labels, bug-tracking identifiers, and code repository paths.

Creating Sessions From Events

To build a higher level picture of engineering workflows, we organize groups of related events into *sessions*. Each session is designed to represent a contiguous block of time where the engineer works on a single task, such as coding or code review. Sessions are intended to represent active workflows, so we consider only front-end events when creating sessions. A session consists of a unique identifier, the ordered list of event identifiers included in the session, the developer behind those events, the union of the artifacts from those events, a start time that corresponds to the time stamp of the first included event, and an end time that corresponds to the time stamp of the last included event.

Figure 1 visualizes how events are combined into sessions. Sessions are created by grouping multiple events into single sessions when they happen on the same day, happen within some time delta of each other (we use 10 min), and have the same task-identifying artifacts (or no task-identifying artifact at all).

Creating Metrics From Sessions

Once events are organized into sessions, those sessions can be used to derive other metrics about developers' behaviors. We selected seven

metrics because they are useful in answering various questions about developer behavior and ones that our events could plausibly capture:

1. Coding time: representing the time spent writing or maintaining code.
2. Reviewing time: representing the time spent reviewing code.
3. Shepherding time: representing the time spent addressing code review feedback.
4. Investigation time: representing the time spent reading documentation.
5. Development time: representing the time spent performing a development activity, of any type.
6. Email time: the time spent interacting with email.
7. Meeting time: the time spent in meetings.

InSession writes data in a format over which we can execute SQL queries, allowing for rapid analyses. For example, the following query examines total daily coding time before and after the declaration of COVID-19 as a global pandemic:

```
SELECT
  date,
  date >= DATE('2020-03-11') as is_during
  _pandemic,
  COUNT(DISTINCT employee) as num_work-
  ing_employees,
  SUM(duration_micros) as total_coding
  _duration_per_day
FROM coding_time
GROUP BY date;
```

Validation Study

To use InSession with confidence in future studies, we performed a validation to understand the extent to which our metrics and

behavioral self-reports of time use agree. Prior research has lamented the lack of validation in similar systems.³

To obtain behavioral reports, we recruited 25 Google engineers to create diaries about what they did for a day, then compared their diaries against our sessions, both qualitatively and quantitatively, using the prevalence and bias adjustments Kappa (PABAK) agreement score,⁶ which ranges from -1 (perfect disagreement) to 1 (perfect agreement).

Figure 2 visualizes PABAK scores. To interpret PABAK scores, we use Allen and Yen's benchmarking approach called *norm-referencing*.⁷ Because email (PABAK = 0.84) and meeting times (0.74) are intuitively the simplest metrics, we treat their empirical agreement as de facto high, then compare all other metrics against this benchmark. By this measure, reviewing time has high agreement (0.81), with coding (0.69) and investigation (0.70) nearly meeting this benchmark. Development time agreement (0.45) is well below this benchmark. (For meeting time, we excluded 10 participants because calendar data were unavailable for the day they completed the study; thus, $N = 15$ for meeting time agreement, $N = 25$ for all other agreement calculations. We return to the issue of missing data in the "Discussion" section.)

To understand mismatches between logs and diaries, we hand-analyzed data from the three participants for each session type who had the worst agreement scores among the 25 participants. Disagreements stemmed from four main sources: participants using tools that we do not yet have logs for, meetings that we did not have events for,

participants multitasking, and participants forgetting to write tasks in their journals.

Overall, we interpret these results to mean that our reviewing, investigation, and coding time metrics are acceptably accurate, but further refinement to the development time metric is needed. The qualitative results suggest improvements to InSession and some fundamental limitations of capturing developer behavior through log data.

Application: The Impact of Readability

We next present a study illustrating the type of analysis that is possible with InSession. In this study, we evaluate the effects of *readability*, a process of programming language certification at Google. Readability certifies that a developer understands best practices and coding style for

a specific programming language. We analyze C++ and Java readability, but readability exists for other languages.

Readability has been in use for many years at Google, but the benefits of readability were not clearly understood. We conducted a mixed-methods study to examine this, using our sessions data to examine two hypotheses, and surveys to provide context to those hypotheses. Our hypotheses are that, after a developer obtains readability certification,

- it will take reviewers less time to review their code
- it will take them less time to respond to reviewers' comments.

These hypotheses are based on the theory that if developers have readability, then their code will have fewer commonly encountered issues

in code, resulting in reviewers not having to point these issues out, and the author not having to acknowledge and fix the issues. To evaluate our hypotheses, we gathered code reviews of changelist authors who went through the readability process in a 10-month window, including their reviews before, during, and after the readability process. This included 104,947 code reviews for C++ and 99,614 code reviews for Java. We ran a linear regression that controlled for developer tenure, number of reviewers, and the size of the change. We additionally included a random effect for author identity to control for characteristics of the individual developer.

The analysis shows the following effects ($p < 0.05$):

- An author having readability is associated with a review time that

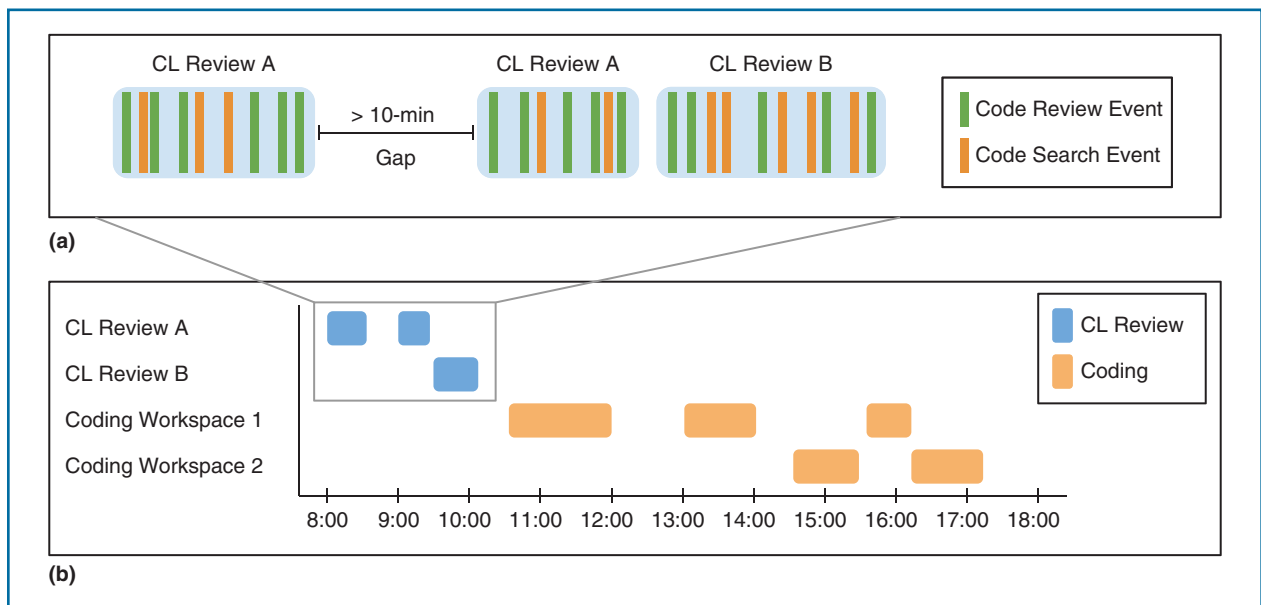


FIGURE 1. (a) This is an example of code review and code search tool events and how they are grouped into three sessions. Based on the changelist (CL) they are associated with and the amount of time between them. (b) This is an example of an engineer's time line of sessions split by task. Gaps can indicate, for example, breaks for lunch, unscheduled discussions, and tool usage for which we have not yet implemented log processing.

is lower by 4.5% for C++ and a nonstatistically significant amount for Java.

- An author having readability is associated with a shepherding

time that is lower by 10.0% for Java and 10.5% for C++.

These quantitative data support the hypotheses that readability has a

positive effect for C++, with some support for Java. Survey data bear this out as well. Eighty-eight percent of engineers that completed the Java readability process said they agree with the

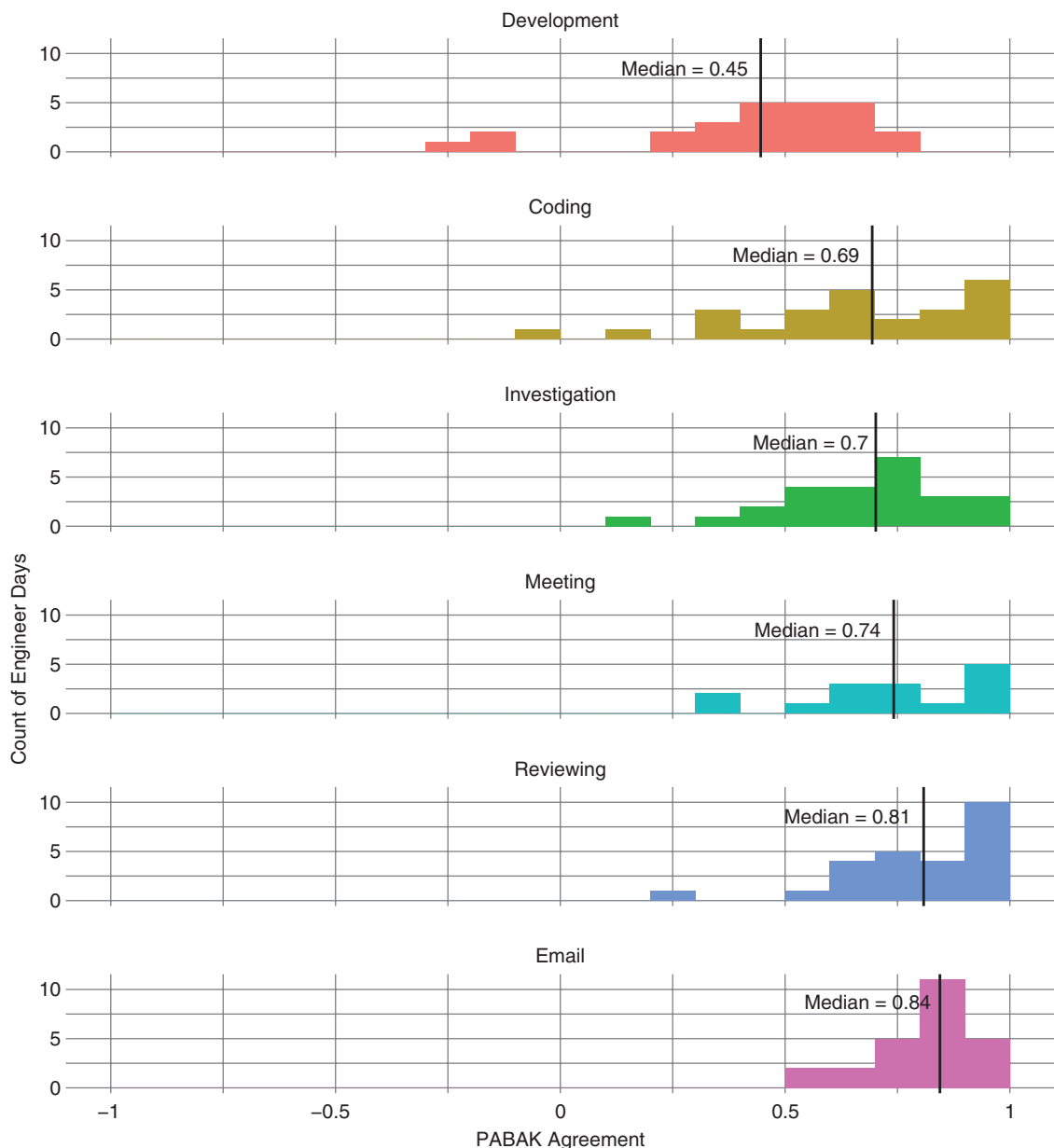


FIGURE 2. The PABAK scores are segmented by activity type. Each data point in each histogram represents all of one participant's activities of that type in their day.



CIERA JASPÁN is software engineer and manager at Google in developer intelligence. Her research interests are empirical software engineering, mining software repositories, program analysis, and productivity metrics. Jaspán received her Ph.D. from Carnegie Mellon University. Further information about her can be found at <https://research.google/people/CieraJaspán/>. Contact her at ciera@google.com.



BEN HOLTZ is a software engineer at Google in developer intelligence. His research focuses on software developer productivity. Holtz received his M.S. in computer science from Stanford University. Contact him at benholtz@google.com.



MATT JORDE is a software engineer at Google in developer intelligence. His research interests include software engineering topics, such as human factors and software quality. Jorde received his M.S. in computer science from the University of Nebraska at Lincoln. He is a member of ACM. Contact him at majorde@google.com.



EDWARD SMITH is a software engineer at Bloomberg. Smith received his B.Sc. in computer science and psychology from the University of Maryland College Park. Contact him at esmith404@bloomberg.net.



CAROLYN EGELMAN is a quantitative user experience researcher at Google in developer intelligence. Her research focuses on software developer productivity. Egelman received her Ph.D. from Carnegie Mellon University. Further information about her can be found at <https://research.google/people/106840/>. Contact her at cegelman@google.com.



MAGGIE HODGES is a user experience researcher with Artech Information Systems at Google in developer intelligence. Her research focuses on software developer productivity. Hodges received her M.S. of public health from the University of California, Berkeley. Contact her at hodgesm@google.com.



COLLIN GREEN is a user experience researcher and manager at Google in developer intelligence. His research focuses on applying combined quantitative and qualitative methods to understand developer experience and engineering productivity. Green received his Ph.D. in psychology from the University of California, Los Angeles. Further information about him can be found at <https://research.google/people/107023/>. Contact him at colling@google.com.



ANDREA KNIGHT is a user experience researcher at Google. Her research interests include product development research, office and engineering productivity, and user privacy. Knight received her M.S. in computer science from Carnegie Mellon University (Human-Computer Interaction Institute). Further information about her can be found at <https://research.google/people/author11433/>. Contact her at aknight@google.com.





LIZ KAMMER is a software engineer at Google. Her research interests are in software engineering, specifically in software developer diversity and inclusion. Kammer received her M.S. in computer science from the University of Alabama. Contact her at eakammer@google.com.



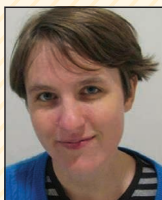
LAN CHENG is a quantitative user experience researcher at Google in developer intelligence. Her research interests are in measurement and impact evaluation of engineering productivity. Cheng received her Ph.D. in economics from the University of California, Davis. She is a member of the Agricultural & Applied Economics Association. Contact her at lancheng@google.com.



JILL DICKER is a software engineer at Google in developer intelligence. Her research interests are in software engineering, specifically in software developer diversity and inclusion. Dicker received her M.Sc. in computer science from Simon Fraser University. Contact her at jdicker@google.com.



MARK CANNING is a software engineer at Google in developer intelligence. His research interests include engineering productivity and predictive modeling. Canning received his B.A. in physics and in mathematics from the University of California, Berkeley. Contact him at argusdusty@google.com.



CAITLIN SADOWSKI is a software engineer and the manager of the Chrome Metrics and Analysis teams at Google. Her research interests include programming languages, software engineering, and human-computer interaction. Sadowski received her Ph.D. from the University of California, Santa Cruz. Further information about her can be found at <https://research.google/people/CaitlinSadowski/>. Contact her at supertri@google.com.



EMERSON MURPHY-HILL is a research scientist at Google in developer intelligence. His research interests include software engineering and human-computer interaction. Murphy-Hill received his Ph.D. in computer science from Portland State University. Further information about him can be found at <https://research.google/people/EmersonMurphyHill/>. Contact him at emersonm@google.com.



JAMES LIN is a software engineer at Google in developer intelligence. Her research interests are in software engineering, specifically in software developer diversity and inclusion. His research interests include end-user programming and user interface design tools. Lin received his Ph.D. in computer science from the University of California, Berkeley. He is a member of ACM and a Member of IEEE. Further information about him can be found at <http://jameslin.name>. Contact him at jameslin@google.com.

statement, “My readability experience was positive overall,” with 87% saying the same about the C++ readability process.

InSession can help investigate other questions about developer behavior. For instance, we recently found that we can predict negative interpersonal interactions during code review by using the 90th percentile of both reviewing and shepherd time.⁸ As another (unpublished) example, InSession helped us show that developers using a new version control system got their changes reviewed more quickly because the new tooling made it easier to create many small changes.

Finally, this study illustrates some of the limitations of InSession. For instance, we must assume that changes before and after the readability process are of equivalent quality; InSession itself provides no data to validate that assumption. Similarly, InSession does not say anything about whether engineers like the process, which is why we must collect complementary survey data.

Discussion

In this article, we have described the implementation, validation, and an application of InSession. Through building and maintaining InSession for more than three years, we have learned the following lessons:

- **Prioritize log sources.** Prioritize the inclusion of log sources based on how useful each source is and on how easily it can be included. We prioritize logs from highly adopted tools and those that are related to our research goals.
- **Enrich data as necessary.** We found that some log sources

benefited by enriching existing data; for example, adding gain and lose focus events to the code review tool logs increased the amount of measurable review activity by about 2 h per week per engineer on average.

- **Validate data and metrics.** We found that the quality and consistency of data can differ between log sources, including unset or nonsensical values for data fields (e.g., negative durations), periods of missing or reduced data due to temporary system instability (e.g., this missing calendar data, described in our validation study), and outlier use cases (e.g., personal automation scripts). We also found that derived metrics, such as development time, while intuitive, can still disagree with human assessments. Thus, we recommend human validation like we describe in this article, but we also recommend automated monitoring and alerting for unexpected data during log ingestion.

While InSession isn’t perfect, we have found it useful to understand developer behavior at scale. We envision a future where similar systems are deployed in other industrial and open source ecosystems, helping answer the most pressing questions about developers’ behavior at work.

References

1. G. Murphy, M. Kersten, and L. Findlater, “How are Java software developers using the Eclipse IDE?” *IEEE Softw.*, vol. 23, no. 4, pp. 76–83, 2006. doi: 10.1109/MS.2006.105.
2. K. Damevski, D. C. Shepherd, J. Schneider, and L. Pollock, “Mining sequences of developer interactions in visual studio for usage smells,” *IEEE Trans. Softw. Eng.*, vol. 43, no. 4, pp. 359–371, 2016. doi: 10.1109/TSE.2016.2592905.
3. P. Johnson, “Requirement and design trade-offs in Hackstat: An in-process software engineering measurement and analysis system,” in *Proc. 1st Int. Symp. Empirical Software Engineering and Measurement (ESEM)*, 2007, pp. 81–90. doi: 10.1109/ESEM.2007.36.
4. A. Sillitti, A. Janes, G. Succi, and T. Vernazza, “Collecting, integrating and analyzing software metrics and personal software process data,” in *Proc. Euromicro Conf.*, Sept. 2003, p. 336–343. doi: 10.1109/EURMIC.2003.1231611.
5. A. Begel and T. Zimmermann, “Analyze this! 145 questions for data scientists in software engineering,” in *Proc. 36th Int. Conf. Software Engineering*, 2014, pp. 12–23. doi: 10.1145/2568225.2568233.
6. T. Byrt, J. Bishop, and J. B. Carlin, “Bias, prevalence and kappa,” *J. Clin. Epidemiol.*, vol. 46, no. 5, pp. 423–429, 1993. doi: 10.1016/0895-4356(93)90018-V.
7. M. Allen and W. Yen, *Introduction to Measurement Theory*. Monterey, CA: Brooks, 1979.
8. C. D. Egelman et al., “Pushback: Characterizing and detecting negative interpersonal interactions in code review,” in *Proc. Int. Conf. Software Engineering*, to be published.



IEEE COMPUTER SOCIETY

DIGITAL LIBRARY

Access all your IEEE Computer Society subscriptions at
computer.org/mysubscriptions