

Organizational Ohm's Law in Practice: Measuring Engineering Productivity

Fei Liu  and Todd McKinnon

Abstract—Engineering productivity measures how effectively engineering teams produce valuable outcomes during a specific time frame. In this article, we aim to quantitatively explore the use of organizational Ohm's law to measure productivity in software engineering. According to this law, organizational productivity is directly proportional to outcome–output efficiency and organizational potential, while inversely proportional to organizational resistance. We derive measurable metrics from this law, which include engineer time allocation, engineer motivation, engineer skills, outcome–output alignment, lines of code, and usage-based business impact. We propose various practical methods to obtain these metrics and explain the considerations to determine the appropriate methods. Using these metrics, we comprehensively study and document four software product development projects. We apply the law to measure the engineering productivity of the projects. Our research supports the validity of organizational Ohm's law using lines of code output and usage-based business impact outcome. Our work demonstrates that organizational Ohm's law could provide an effective and intuitive model for understanding engineering productivity challenges.

Index Terms—Alignment, artificial intelligence (AI), business impact, engineering productivity, engineering velocity, lines of code (LOC), motivation, organizational Ohm's law (OOL), outcome, output, productivity, skill, time allocation.

I. INTRODUCTION

SOFTWARE engineering productivity research has gained great attention from researchers and practitioners with diverse backgrounds over the past half-century [1], [2], [3], [4], [5], [6], [7], [8], [9]. However, the quest to understand software development and improve its productivity continues. The ongoing interest in software engineering productivity is partially driven by enormous demand: software is eating the world [10], and every company has become a software company [11]. This interest is also driven by a lack of supply: the goal of software development has expanded from delivering high-quality products on time and within budget to delivering high-value products that delight customers. Software technology, tools, development methodology, and business approaches have been explored and improved constantly. However, there are still significant gaps in equipping engineers and managers with easy-to-use and comprehensive engineering management models so that they can make informed decisions.

Manuscript received 2 February 2024; revised 23 March 2024; accepted 25 June 2024. Date of publication 3 July 2024; date of current version 18 July 2024. Review of this manuscript was arranged by Department Editor M. Dabic. (Corresponding author: Fei Liu.)

The authors are with the Okta Inc., San Francisco, CA 94105 USA (e-mail: fei.liu@okta.com; tmckinnon@okta.com).

Digital Object Identifier 10.1109/TEM.2024.3421892

Early software engineering productivity studies focus on sharing insightful qualitative empirical observations for software engineering. For example, Conway's law describes the relationship between a system and its design organization and concludes that a design effort should be organized according to communication needs [1]. The mythical man-month points out that adding manpower to a late software project makes it slower due to increased intercommunication overhead [2].


Researchers in the 80s and 90s made significant progress in quantitatively modeling software engineering planning and management. Software cost estimation methods, such as the constructive cost model (COCOMO) [12], [13] and function point analysis [14], [15], have been developed to a high level of detail. However, the cost estimation methods cannot be meaningfully applied without extensive initial calibrations to company-specific local data. In the meantime, a balanced scorecard was introduced from a strategy and general management angle. It encourages goal setting and performance measurement with a balanced scorecard. Even though the work is not software engineering-focused research, it points out the importance of looking at (engineering) business not only from an internal (engineering) business perspective but also from financial, customer, innovation, and learning perspectives [16], [17].

Recent software development and management studies and practices, such as story points [18], scaling agile@Spotify [19], and the two-pizza rule [3], are developer centric and built on DevOps and agile concepts. The industry software productivity or engineering velocity best practices are practical, valuable, and widely adopted, but typically only focus on particular aspects of productivity and are hard to prove scientifically and learn by other organizations [20]. Over the past several years, studies, such as DevOps metrics [5], [6] and a SPACE method [8], emphasize that developer productivity is influenced by multidimensional input metrics, such as developer satisfaction and well-being. However, the studies do not connect the input metrics with a formulable relationship to help understand their inter-relationship.

Table I presents a comparison of research studies on software engineering productivity. These studies address various aspects of software engineering planning and management, and use different methodologies. The comparison evaluates whether the research studies provide qualitative insights or quantitative models, focus on engineering or business impact, and consider multidimensional inputs beyond software development delivery. However, none of the research studies provide a model that can meet all five criteria. In practice, dominant software

TABLE I
COMPARISON OF SOFTWARE ENGINEERING PRODUCTIVITY RESEARCH WORK

Research Work	Qualitative Insight	Quantitative Model	Engineering Impact	Business Impact	Multi-dimensional inputs
Conway's law [1]	✓	○	✓	x	x
Mythical man-month [2]	✓	○	✓	x	x
COCOMO and COCOMO II [12],[13]	✓	●	✓	x	x
Function point analysis [14],[15]	✓	●	✓	x	x
Balanced scorecard [16],[17]	✓	◐	✓	✓	x
Story point analysis [18]	✓	●	✓	x	x
DevOps metrics [5],[6]	✓	◐	✓	x	✓
SPACE method [8]	✓	◐	✓	x	✓
OOL [22, this work]	✓	●	✓	✓	✓


 ○ proposed metrics
 ● developed formulae

development management approaches are still based on expert judgment within silos, despite the fact that less tech-centric functions, such as sales, marketing, finance, and human resources, use data-driven management approaches. Therefore, there is growing interest in continually developing a practical and measurable engineering productivity model.

Moreover, the engineering management community still lacks systemic, real-world empirical studies on engineering productivity. This is because collecting empirical data requires significant operational effort. It can also involve some form of employee monitoring. Hence, careful consideration is necessary to choose suitable approaches to obtain the data and use the engineering productivity process as a learning and improving mechanism for engineers and managers. Although some industry case studies have been collected to support traditional software cost estimation methods [12], [21], technological capabilities, engineering practices, business expectations, and culture for how to build valuable software products have all changed dramatically over the past few decades. There is a need to document modern software engineering case studies with multidimensional inputs quantitatively.

Furthermore, organizational Ohm's law (OOL) was proposed to describe the relationship between organizational productivity (also known as organizational velocity) and its influencing factors: outcome–output efficiency, organizational potential, and organizational resistance [22]. Qualitative reasoning and empirical examples were provided to illustrate the comprehensiveness and intuitiveness of the model. However, it is interesting to demonstrate the quantitative nature of OOL. It is also valuable to apply OOL to modern software development to show its usefulness or limitations.

In this work, we develop an easy-to-use quantitative engineering productivity model: quantitative OOL. The model can be used for multidimensional outcomes. It is not only helpful in measuring engineering-focused metrics, such as lines of code (LOC), but also applicable to evaluating business-focused metrics, such as business impact. The model takes into account multidimensional inputs, including software development processes (percentage of time spent on coding, documentation, company processes, engineering processes, testing, and infrastructure), organizational structure and communication (percentage of time spent on communicating or decision waiting), employee

growth (engineering motivation and skills), and goal alignment (outcome–output alignment). We apply the quantitative OOL to industry product development projects and document the project details and their engineering productivity metrics. We discuss considerations required to choose suitable approaches to obtain engineering productivity metrics. The work demonstrates the ways in which engineers and managers can use quantitative OOL in their day-to-day work.

The rest of this article is organized as follows. In Section II, we will explain our research methodology. In Section III, we will derive a quantitative OOL equation and identify engineering productivity metrics. In Section IV, we will apply OOL quantitatively to analyze product feature developments and test the validity of OOL. In Section V, we will discuss the limitations of the work and future directions. In Section VI, we will conclude this article.

II. METHODOLOGY

Fig. 1 provides a schematic overview of the methodology employed. First, we start with OOL, inspired by Ohm's law in electrical engineering, to formulate a quantitative OOL equation. We identify metrics that can be measured objectively or subjectively to represent the variables in the quantitative OOL. Second, we choose four software product development projects to study and document the details of the projects. Third, we apply the quantitative OOL to the selected projects. We evaluate various approaches to collect the OOL metrics, discuss their pros and cons, and choose the most suitable approaches to obtain them. For OOL input metrics, we obtain organizational resistance metrics in the form of time allocation percentage, and individual potential metrics in the form of engineer skill and motivation scores. For OOL output and outcome metrics, we obtain output metrics in the form of lines of code, and outcome metrics in the form of business impact. We also obtain project duration and outcome-output efficiency metrics for lines of code output and business impact outcome. Fourth, we use the collected metrics to test the validity of OOL under lines of code output and business impact outcome scenarios. In summary, we derive a quantitative OOL equation and use a quantitative deductive approach with data collected from cross-sectional case studies to test its validity in this work [23].

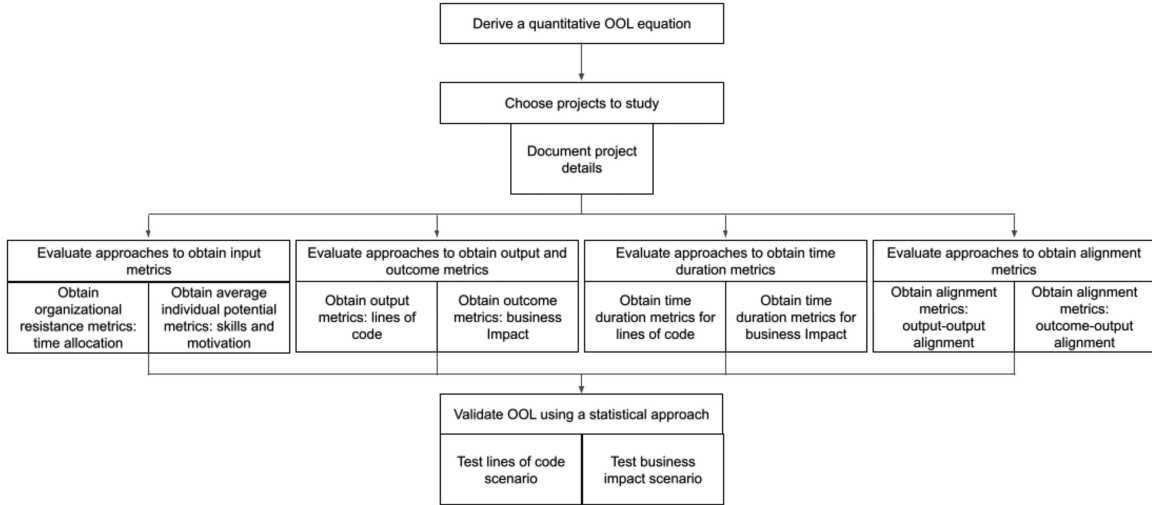


Fig. 1. Research method diagram.

III. MAKING OOL MEASURABLE

OOL, inspired by Ohm's law in electrical engineering, is introduced to describe organizational productivity, i.e., organizational velocity. OOL states that outcome current, and hence organizational velocity, is proportional to outcome–output efficiency and organizational potential, and inversely proportional to organizational resistance [22]

$$\begin{aligned}
 I_{\text{outcome}} &= \eta \times I_{\text{output}} = \eta \times \frac{V}{R} \\
 \frac{O}{D} &= \eta \times \frac{\bar{v}}{R^{\text{structure}} + R^{\text{process}} + R^{\text{infrastructure}}} \\
 &= \eta \times \frac{\bar{v}}{R^{\text{self}} + R^{\text{collaboration}} + R^{\text{process}} + R^{\text{infrastructure}}}
 \end{aligned} \quad (1)$$

where I_{outcome} is the outcome current, I_{output} is the output current, O is an outcome, D is the time duration to achieve the outcome, η is the outcome–output efficiency, V is the organizational potential, R is the organizational resistance, \bar{v} is the average individual potential, $R^{\text{structure}}$ is the structure resistant, R^{self} is the self-resistance, $R^{\text{collaboration}}$ is the collaboration resistance, $R^{\text{infrastructure}}$ is the infrastructure resistance, and R^{process} is the process resistance.

We use outcome current to describe organizational velocity in terms of an effective outcome. An outcome is the net useful output that aligns with organizational goals. Output describes deliverables produced by organizations. An outcome depends on the nature of the organizations and their objectives. For example, engineering, product, and design organizations often aim to provide customers with the features they need and want. Output that aligns with the objectives of the organization is treated as an outcome. The ratio between outcome and output is outcome–output efficiency.

Organizational resistance is a summation of structure resistance, process resistance, and infrastructure resistance. Structure resistance consists of self-resistance and collaboration resistance. Similar to intrinsic resistance from copper wire in the engineering setting, self-resistance is the minimum resistance

engineers impose on an organization. Self-resistance can be thought of as resistance that assumes that individuals can work without collaboration overhead and use optimal processes and infrastructure. Collaboration resistance is the resistance due to project communication and dependencies [24]. Process resistance is the resistance to following engineering and company processes, and infrastructure resistance is the resistance due to inefficient infrastructure.

OOL and its equivalent circuit approach are applied qualitatively to explain and optimize various engineering productivity situations [22]. However, to obtain a practical quantitative OOL, we need to derive easy-to-obtain metrics for its variables. When formulating a quantitative OOL equation, we aim to prioritize simplicity so that OOL can be usable in real-world contexts

$$\begin{aligned}
 \frac{O}{D} &= \eta \times \frac{\bar{v}}{R^{\text{self}} + R^{\text{collaboration}} + R^{\text{process}} + R^{\text{infrastructure}}} \\
 &= \frac{\eta}{R^{\text{self}}} \times \frac{\bar{v}}{R^{\text{self}} + R^{\text{collaboration}} + R^{\text{process}} + R^{\text{infrastructure}}} \\
 &\sim \frac{\eta}{R^{\text{self}}} \times \frac{\bar{v}}{\frac{T^{\text{self}} + T^{\text{collaboration}} + T^{\text{process}} + T^{\text{infrastructure}}}{T^{\text{self}}}} \\
 &\sim \frac{1}{r^{\text{self}}} \times \bar{N} \times \eta \times \frac{\bar{v}_{\text{max}} \times \bar{v}}{\bar{v}_{\text{max}}} \times \frac{T^{\text{self}}}{T} \\
 &\sim \frac{\bar{v}_{\text{max}}}{r^{\text{self}}} \times \bar{N} \times \eta \times \frac{\bar{m}}{\bar{m}_{\text{max}}} \times \frac{\bar{s}}{\bar{s}_{\text{max}}} \times \frac{T^{\text{self}}}{T} \\
 &\sim k \times \bar{N} \times \eta \times \frac{\bar{m}}{\bar{m}_{\text{max}}} \times \frac{\bar{s}}{\bar{s}_{\text{max}}} \times \frac{T^{\text{self}}}{T}
 \end{aligned} \quad (2)$$

where T^{self} is the time spent for self-engineering development, $T^{\text{collaboration}}$ is the time spent due to collaboration, T^{process} is the time spent for processes, $T^{\text{infrastructure}}$ is the time wasted on inefficient infrastructure, T is the working time, r^{self} is the average individual self-resistance, \bar{N} is the average number of engineers working on the project during the project duration, \bar{v}_{max} is the maximum average individual potential, \bar{m} is the average engineer motivation, \bar{s} is the average engineer skills, \bar{m}_{max} is the maximum average engineer motivation, \bar{s}_{max} is

the maximum average engineer skills, and k is a constant with its value equal to the maximum average individual potential dividing by average individual self-resistance.

Equation (2) transforms the abstract idea of organizational resistance into quantifiable time allocation measurements. We use time allocation as a measurable metric to represent the components of organizational resistance because time allocation is relatively easy to obtain through surveys or information technology tools and is widely used by organizations to study productivity [25], [26]. It is also because the concept of time spent can be intuitively connected with the concept of resistance.

The derivation assumes that the first-order approximation of the average individual self-resistance is a constant across projects, which is because the individual self-resistance is a hypothetical limitation and an average across a group of engineers. However, the second-order effect of average individual self-resistance may depend on the nature of software development, such as software use cases, product development stages, software architecture, and programming languages [13], [27], [28].

Average individual potential is a function of average engineer skills and motivation. The derivation makes a simplification and uses the product of average engineer skills and motivation as an approximation of average individual potential. We also introduce the maximum average individual potential, maximum average engineer motivation, and maximum average engineer skills to scale average individual potential, average engineer motivation, and average engineer skills, respectively, so that the metrics can be easily achieved and analyzed using surveys.

Equation (2) states that an outcome during a period is proportional to an average number of engineers working on a project (\bar{N} , with a unit of people), alignment with a desired goal (η , a relative parameter without unit), relative average engineer motivation and skills ($\frac{\bar{m}}{\bar{m}_{\max}}$, $\frac{\bar{s}}{\bar{s}_{\max}}$, relative parameters without units) and percentage of time spent on core engineering development ($\frac{T^{\text{self}}}{T}$, a relative parameter without unit), and a constant ($k = \frac{\bar{v}_{\max}}{r_{\text{self}}}$). The constant k is a hypothetical limitation of individual engineering productivity; its unit is the unit of outcome metric per person per time.

In contrast to Ohm's law in electrical engineering, where variables are independent, the input variables on the right side of (2) may be interdependent. For instance, if more engineers are added to a project, it would increase the time required for the team to collaborate, resulting in a decrease in the percentage of time spent on core engineering development. Eliminating engineering or company processes would increase the percentage of time spent on core engineering development but could pose risks in terms of aligning a project with its goal or motivating and upskilling engineers.

IV. APPLYING OOL TO PRODUCT-FEATURE DEVELOPMENT

In this section, we will apply OOL in real-world product development situations. We will explain the projects selected for the study, the approaches to obtain measurable metrics for the variables in OOL, and the validation of OOL.

A. Choosing Projects to Study

To begin designing measurements, it is important to decide which projects to evaluate. Okta, an enterprise software company, has an annual innovation survey process. In the process, the engineering and product leadership nominates the top projects their team delivers throughout the year, and employees in the sales organization are invited to rate the nominations. In this study, we utilize the company-wide survey process by selecting product-feature development projects from the nominations. These projects are significant software product development that can directly affect the company's overall performance, and some may even be considered new products. From a managerial perspective, keeping track of productivity metrics allows the company to gain timely engineering management insights. From a research perspective, these projects represent large-scale, modern industry software product development. We can gather valuable datasets, conduct systematic studies, and gain practical insights by examining these projects.

We selected the four most significant product-feature development projects. These features are detailed in Table II. Project A is an IT/security feature that can be used by employees, contractors, or partners. It has both front-end user interfaces and back-end functions, and is compatible with major operating systems. The backend is written in Java, while the frontend is written in JavaScript. Its macOS/iOS clients are written in Object C and Swift, while its Android clients are written in Kotlin and Java. Its Windows clients are written in C/C++. Most of the codes were built on the company's core monolith repository, which has more than ten million LOC. Project B is also an IT/security feature, but it is used by administrators. It has a back-end function and front-end user interfaces, and is written using Java and JavaScript. It is also built on top of the core monolith repository. Project C is a no-code platform as a service. It is an IT/security feature used by administrators. It is written using node.js and Rust. It utilizes a microservices architecture. Project D is an IT/security feature used by administrators with end-user participation. It consists of back-end functions and front-end user interfaces. It is written using Java and node.js. The project is built using a macroservices architecture, which consists of microservices within the core monolith with shared databases. These projects took from 13–28 months from the projects' start to the general availability (GA) of the product features. The average number of engineers per team varies from 5.3 to 38. Project C uses a tuck-in acquisition to speed up product development. The tuck-in mergers and acquisitions (M&A) is a process in which a larger company acquires a smaller one to integrate the newly acquired technical capability into its own platform.

These selected product features have been developed over multiple years as a part of enterprise software-as-a-service IT/security products. They cover a wide range of complex software development situations, such as complicated software requirements, multiple programming languages, mixed software architecture, and the choice of M&A.

TABLE II
PROJECT DETAILS AND OOL MEASURABLE METRICS

Metric		Project A	Project B	Project C	Project D	
Project Information	Function	Backend, frontend, clients	Backend, frontend	Integrations, frontend	Backend, frontend	
	Language	Java, JavaScript, Swift, Kotlin, Objective C, C/C+	Java	Node.js, Rust	Java, Node.js	
	Architecture	monolith	monolith	microservices	macroservices, microservices	
Project Schedule	Project duration in month	28.00	17.00	13.00	21.00	
	Average number of engineers	38.00	5.30	18.00	15.29	
Project Resource	Whether to integrate with an acquisition	No	No	Yes	No	
Input Metric	Percentage of time spent on communicating or decision waiting	9%	5%	18%	5%	
	Percentage of time spent on coding	42%	45%	53%	55%	
	Percentage of time spent on documentation	5%	5%	5%	5%	
	Percentage of time spent on company processes	5%	4%	7%	5%	
	Percentage of time spent on engineering processes	9%	6%	5%	5%	
	Percentage of time spent on testing	8%	30%	5%	10%	
	Percentage of time wasted in infrastructure inefficiency	8%	3%	2%	10%	
	Engineer Motivation	Relative motivation score	0.74	0.80	0.68	0.97
	Engineer Skill	Relative skill score	0.52	1.00	0.68	0.77
Output Scenario	LOC written from project start to GA	453 357.00	144 104.00	140 618.00	313 013.00	
	LOC GA	595 222.00	149 011.00	5 842 873.00	313 013.00	
	LOC per engineer per year	5113.00	19 193.00	7211.00	11 701.00	
	Outcome-output Efficiency	Output-output alignment score	1.00	1.00	1.00	1.00
	Time Duration	Equivalent project duration in month	28.00	17.00	13.00	21.00
	OOL	Output current per year	194 296.00	101 720.00	129 801.00	178 865.00
Outcome Scenario	OOL	Input product	7.33	3.18	4.79	7.45
	Usage-based Business Impact	Usage-based business impact	13 105.91	2 002.59	14 199.40	8095.46
	Outcome-output Efficiency	Outcome-output alignment score	0.89	0.77	0.83	0.77
	Time Duration	Equivalent project duration in month	28.00	17.00	49.00	21.00
	Project Performance	Project performance	0.17	0.46	0.22	0.38
	OOL	Outcome current per year	5616.82	1413.59	3477.41	5596.93
	Alignment*Input product	6.54	2.46	3.96	5.73	

B. How to Measure OOL Input Metrics

1) *Evaluating Approaches to Obtain Input Metrics:* According to (2), organizational resistance can be measured by analyzing engineer time allocation, and individual potential can be

determined by evaluating relative average engineer motivation and skills. One method involves using modern hardware and software, such as laptops, mobile devices, workstations, Google or Microsoft calendars, Slack, Zoom, and Microsoft Teams, to

obtain screen time reports, time breakdowns for meetings and nonmeetings, and track communication time. Git software, such as GitHub or GitLab, records commit activities, which can be used to estimate coding time [29]. Project management tools, such as Jira, can be analyzed to get task info, derive story points, or obtain time spent on tasks and wait time for dependencies [30]. These methods are objective, data-driven, and can be automated, although implementing employee monitoring requires careful planning and transparent communication. The data collection process also requires IT support, and it is impossible to obtain historical data. Therefore, these methods are suitable for continuous engineering velocity operations but may not be practical for research studies or pilots.

Another approach is to collect survey data from engineers. An engineering survey is a convenient tool to track engineering velocity operations. Metrics, such as employee time allocation, satisfaction scores, turnover rates, management ratings, culture scores, and performance scores, can be used to come up with organizational resistance and individual potential. However, a self-reported engineering survey involves reaching out to at least 78 engineers, which requires planned communication and can create an unnecessary burden on their core engineering function. Moreover, some engineers on a project may leave the company at the time of a survey, while others join during the project. Similar to the software usage methods, the engineering survey is not the best way to conduct a backward-looking research study.

Fortunately, the engineering leaders for the four projects studied have been responsible for their projects from the beginning. Also, improving engineering velocity is their responsibility and interest. After evaluating the options, we decided to invite the leaders to fill out an engineering velocity metrics questionnaire.

To ensure information reliability, more than one person was involved in collecting metric data or reviewing it for each project. Typically, other managers, the most senior engineers, or technical project managers were brought into the process. In fact, these leaders often proactively involved additional people to contribute to the data collection.

Initially, we were concerned that the leaders might be unwilling to share information about their team's performance and afraid the data would be used to compare them with peers. So, we transparently communicated the purpose of data collection: data collection is mainly for engineering velocity research, data from multiple projects are collected, and useful learnings may be used to improve engineering operations. Furthermore, the leaders have the most practical knowledge of engineering productivity. Therefore, we not only involved them in filling out the questionnaire but also observed their reactions and encouraged them to ask questions and provide suggestions. Surprisingly, they welcomed the productivity study and took it as an opportunity to share engineering velocity observations and challenges.

2) *Obtaining Organizational Resistance and Average Individual Potential Metrics:* We asked the leaders to provide information about engineer time allocation, engineer motivation, and engineer skills, as shown in the corresponding rows of Table II. For time allocation, we asked leaders to specify the percentage of time their teams spent on different activities, including communication or waiting for decisions, waiting for dependencies,

coding, documentation, doing company processes, doing engineering processes, testing, and dealing with the inefficiency of engineering infrastructure. Furthermore, we asked the leaders to rate average engineer motivation on a scale of 0–5, with 5 being highly motivated. Also, we requested the leaders to rate the average skills of their teams on a scale of 0–5, with 5 being highly skilled. We suggested that they consider both technical capabilities, such as coding, testing, architecture design, and security, and nontechnical capabilities, such as collaboration, communication, and time management needed for the engineer roles. The subjective skill and motivation scores are similar to employee evaluation, and the leaders feel it is easy to provide the assessment. We then divided the motivation and skill scores by 5 to obtain relative motivation and skill scores.

For each project, we collected metrics during each product-feature milestone, such as at the start of the project, up to the early access (EA) stage, then up to the GA stage, and finally, up to the date of the questionnaire. These metrics help illustrate the software development lifecycle properties and reflect each project's unique hurdles at different stages. This exercise also allows leaders to back up their numbers with practical reasons. However, for the sake of simplicity, we report the time average of the metrics for the four projects, as summarized in Table II.

Interestingly, although we did not explain quantitative OOL and the meaning of the input metrics to the leaders, they could instinctively make connections between the metrics and their team's productivity or the intrinsic properties of their projects. They were able to proactively apply these metrics to explain their situations and come up with ideas to improve their team's performance through the exercise.

C. How to Measure OOL Output and Outcome Metrics

1) *Evaluating Approaches and Measuring LOC Output:* Let us first evaluate the output of software development projects. One commonly used metric is LOC. While LOC-based measurement has many limitations, it is an easy-to-obtain objective measure and can be referenced in project management literature. Therefore, we include LOC as an output metric in this study.

To collect historical LOC data, we asked engineering leaders to extract the information from Git repositories at various project stages, including project start, feature EA stages, GA stages, and the dates of the input metrics questionnaire collected. Some of the project code is a part of core components shared by multiple projects or coexists with the code of other projects. The leaders employed an approximate approach to count LOC belonging to their projects and contributed by their teams. Fig. 2 shows the LOCs of the four features at the three stages of the product-feature lifecycles: project start, EA, and GA. Note that all the engineering teams continue to develop these features after GA. However, for the sake of simplicity, Fig. 2 does not include the data after GA.

To avoid bias, we collected LOC information months after obtaining input metrics, and we knew that the teams had not tracked the project LOC before this research. We also considered different definitions of LOC. Although source LOC is the most widely used definition in the literature, we decided to include

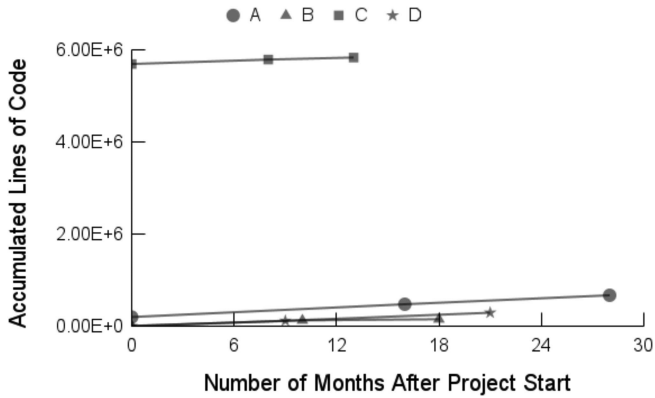


Fig. 2. Accumulated LOC of projects A, B, C, and D at the start of the project, EA stages, and GA stages.

source code, test code, and comments when counting LOC while excluding white spaces. The reason is that test code is often intertwined with source code, and differentiating them is unreliable. More importantly, reliability and security are top priorities for enterprise IT/security products, and the engineering teams put significant effort into designing tests and ensuring high product quality [13]. We also include comments in LOC since they only contribute to a small percentage of the total.

Table II summarizes the LOC added from the start of the project to GA, LOC at the end of GA, and LOC per engineer per year for each project. The projects' LOC per engineer per year falls within the reported range [27]. Project B has the highest LOC per engineer per year due to a large percentage of test code. Project C has reasonable LOC per engineer per year despite its large Git repository size, likely due to its use of microservices architecture. Similarly, Project D has a relatively high LOC per engineer per year number, attributed to its use of macroservices architecture. Microservices and macroservices' architectures decrease coupling with the core monolith, thereby reducing communication and coordination with other teams. At the same time, they result in more LOC to realize business logic and databases due to self-contained software components. Both effects could increase the LOC per engineer per year for the projects.

2) *Evaluating Approaches and Measuring Business Impact Outcome:* Measuring the business impact of product features can be done by looking at the feature's contribution to the company's revenue over its lifetime. However, not all product features have their own unique stock keeping units. Some are packaged as a part of pricing bundles, and some are discounted for specific contracts. These make it challenging to separate intrinsic individual feature revenue contributions. Another way to obtain feature revenue contribution is by multiplying the amount of feature usage by its equivalent unit price, i.e., revenue equal to the number of units multiplied by unit price. This approach has several advantages. First, it provides a relatively objective metric. Second, historical and real-time feature usage data can be obtained objectively from product system logs. Third, building features that customers find valuable to use emphasizes a customer-centric view and creates long-term

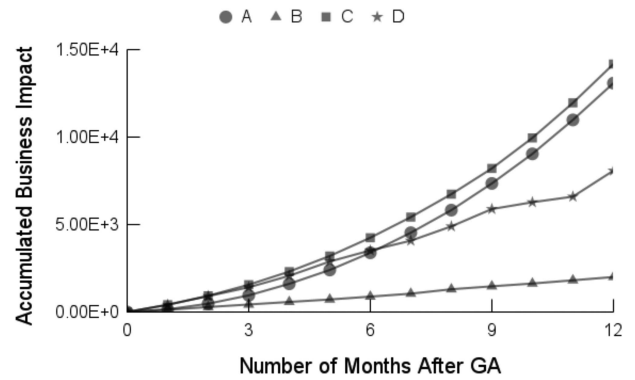


Fig. 3. Normalized discounted accumulated business impact of projects A, B, C, and D over one year after GA.

business value. Fourth, it solves product pricing bundling issues. Additionally, it may align with a usage-based pricing model.

To represent product-feature usage, we obtained the number of organizations that use the features. According to the diffusion of innovation theory [31] and Bass model [32], it is critical to establish and nurture the innovator and imitator relationship at a new product introduction stage. We focused on an organization count instead of an individual user count. The reason is that the innovator and imitator relationship exists mainly at the organizational level for the enterprise business-to-business IT and security product features we study. Organizations, not individuals, make the purchase and implementation decisions. The number of organizations could be a suitable metric to evaluate new feature introduction, while the number of users could be an appropriate metric for assessing mature features.

We obtained each project's accumulated business impact over the one year after GA. Then, we discounted the value to the start dates of the projects using an annual discount rate of 12%. We plotted the normalized discounted accumulated business impact (in units of the number of organizations*\$) for the four features in Fig. 3. In short, we use the normalized discounted accumulated business impact one year after GA as the outcome metric, as shown in the usage-based business impact row of Table II.

D. How to Determine Time Duration to Achieve Outcomes

In this section, we will discuss how to measure time duration to achieve outcomes. To determine the time duration, we need to consider the type of outcomes we aim for.

1) *Determining Time Duration to Achieve LOC Output:* When determining the time duration to achieve LOC output, there is a direct relationship between LOC and the time it takes to write the LOC. Therefore, the time duration it takes to achieve the LOC at GA stages is equal to the project duration between the start of projects and their GA dates.

2) *Evaluating Approaches and Determining Time Duration to Achieve Business Impact Outcome:* When determining the time duration required to achieve a business impact outcome, we need to take into account the project's development process. For Projects A, B, and D, which were developed entirely internally,

the time duration to achieve business impact outcomes is equal to the project duration between the start of the projects and GA dates.

However, Project C was developed with the help of an acquisition, which aimed to accelerate product-feature development. Therefore, to determine the time duration it takes to achieve the business impact outcome, we need to add the internal development time to the time saved due to the acquisition. To determine the amount of development time saved by the acquisition, we requested an estimate from the CEO of the acquired company, who was also the project leader for Project C. We also gathered estimates from the people in product and corporate development teams who made the acquisition decisions. And the individuals who provided estimates also either led or participated in the post acquisition product integration. These estimates were then averaged to produce the time saved due to the acquisition. We then added the time saved to the project duration to get the time duration for the project, as shown in the outcome equivalent project duration row in Table II.

E. How to Determine Outcome–Output Efficiency

In this section, we will discuss how to measure outcome–output efficiency.

1) *Determining Outcome–Output Efficiency of LOC*: For the outcome–output efficiency of LOC, since the output of software engineering is to produce quality code, LOC output can be thought of as a special-case outcome with an outcome–output alignment equal to output–output alignment, with a value of 1.

2) *Evaluating Approaches and Measuring Outcome–Output Efficiency of Business Impact Outcome*: Now, let us measure the outcome–output efficiency for the business impact outcome. For the software product features we study, their goal is to deliver product features that customers love and generate the best business value. The business impact outcome metric we discussed in Section IV-C2 is a representation of the business goal.

The sales team has the most direct interactions with customers and can provide the most valuable insights into customer satisfaction and the perceived business value of product features. We use their feedback as the voice of the customers to measure the alignment metric. As a part of the company's annual innovation survey process mentioned in Section IV-A, the CEO sent survey invitations to sales team members with customer-facing roles. In the survey, participants were asked to rate the nominated projects on a scale of 0–5, with 5 being best loved by customers and generating the most business value. The order of the nominated projects in the survey was randomized to avoid bias. Participants were also given the opportunity to provide detailed comments about the projects or general suggestions on product innovation. We collected over 650 survey responses to calculate the weighted average ratings of the four projects. We then normalized the ratings to a scale of 0–1 and used them as the alignment scores in this study. These scores are shown in the outcome–output alignment row of Table II.

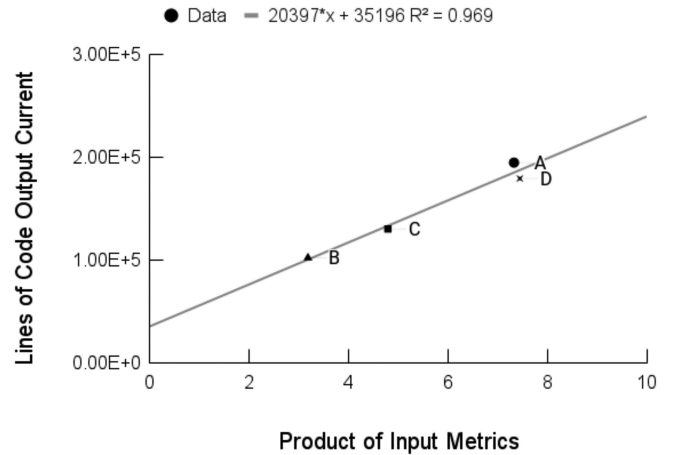


Fig. 4. LOC output current versus product of input metrics of the output scenario for projects A, B, C, and D.

TABLE III
OOL REGRESSION RESULTS

Regression: LOC Output						
	Coef	Std err	<i>t</i>	<i>P</i> > <i>t</i>	[0.025	0.975]
<i>c</i>	3.52e+04	1.54e+04	2.281	0.150	−3.12e+04	1.02e+05
<i>k</i>	2.04e+04	2.59e+03	7.882	0.016	9.26e+03	3.15e+04
Regression: Business Impact Outcome						
	Coef	Std err	<i>t</i>	<i>P</i> > <i>t</i>	[0.025	0.975]
<i>c</i>	−7.81e+02	5.36e+02	−1.456	0.283	−3.09e+03	1.53e+03
<i>k</i>	9.77e+02	1.09e+02	8.989	0.012	5.09e+02	1.44e+03

F. Validating OOL

We have obtained the values for all variables in the quantitative OOL equation for the four projects. As derived in (2), there is a linear relationship between the outcome current metrics and the product of the input metrics and the alignment metrics. In this section, we will test the linearity statistically to determine the validity of OOL.

1) *Testing LOC Scenario*: To calculate output current, we divided LOC by project duration, as shown in the output current per year row in Table II. The numbers represent the left hand of (2). We multiply the percentage of time spent on coding and testing, relative motivation scores, and relative skill scores, as shown in the input product of the output scenario row in Table II. The numbers represent the right hand of (2), except the constant *k*.

We plotted the output current and the product of input metrics of the output scenario for the four projects, as shown in Fig. 4. The data points fit with linear regression with a coefficient *k* of 2.04E4 (in units of LOC per engineer per year), a constant *c* of 3.52E4, and an *R*-square of 0.969. The *p*-value of *k* is 0.016, as shown in Table III, indicating that the linear regression is statistically significant at a significance level of 0.05. The statistically significant linear relationship between LOC output

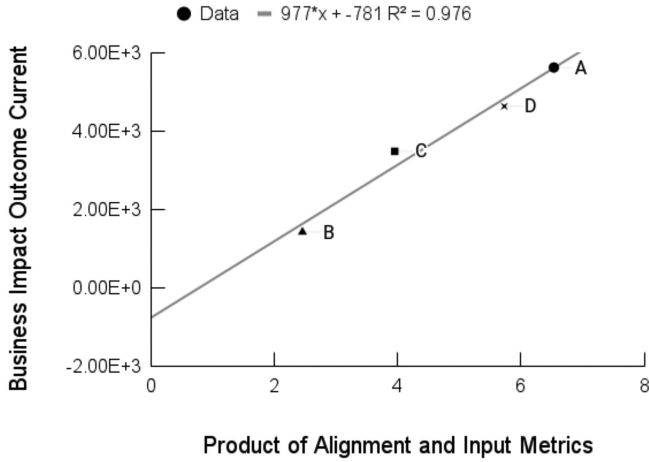


Fig. 5. Business impact outcome current versus product of alignment and input metrics of the outcome scenario for projects A, B, C, and D.

productivity and their corresponding input products supports the validity of OOL.

2) *Testing Business Impact Scenario*: To calculate the outcome current, we divided business impact by equivalent project duration, as shown in the outcome current row per year in Table II. The numbers represent the left-hand side of (2). We multiply outcome–output alignment scores, the percentage of time spent on coding and testing, engineer motivation scores, and engineer skill scores, as shown in the alignment*input product of the outcome scenario row in Table II. The numbers represent the right-hand side of (2), except the constant k .

We plotted the outcome current and the product of alignment and input metrics (alignment*input metrics) of the outcome for the four projects, as shown in Fig. 5. The data points fit with linear regression with a coefficient k of $9.77E2$ (in units of number of organizations*\$ per engineer per year), a constant c of $-7.81E2$, and an R -square of 0.976 . The p -value of k is 0.012 , as shown in Table III, indicating that the linear regression is statistically significant at a significance level of 0.05 . The statistically significant linear relationship between business impact outcome productivity and their corresponding alignment and input products supports the validity of OOL.

3) *Insights From OOL*: As discussed in Section III, the constant k is the reciprocal of average individual self-resistance at the maximum individual potential, representing a hypothetical limitation of individual engineering productivity. For the LOC output, the regression suggests a k of $2.04E4$ (in the unit of LOC per engineer per year), consistent with the upper measure of LOC per engineer per year [27]. For the business impact outcome, the regression suggests a k of $9.77E2$ (in units of the number of organizations*\$ per engineer per year). Hence, the linear regression of the projects not only supports the validity of OOL but also provides a way to obtain the limitation of individual engineering productivity.

Next, let us compare the alignment and input metrics of the four projects. We can see that Project A was well aligned with the business outcome. However, the team may need to improve the skills of its engineers, better manage dependencies, and

streamline the engineering process. Project B had skillful engineers but could build the feature to align better with the business outcome. Project C aligned well with the business outcome, but the engineers needed better motivation and could shorten the time spent on communication, waiting for dependencies, and completing company processes, which may suggest a need to optimize the M&A integration process. Project D had motivated engineers but could build the feature to align better with the business outcome.

Moreover, the OOL can be interpreted in another way. As shown in (2), it states that the outcome current is proportional to the maximum achievable individual outcome current for the organization (k), the average number of available engineer resources (\bar{N}), and project performance ($\eta \times \frac{\bar{m}}{\bar{m}_{\max}} \times \frac{\bar{s}}{\bar{s}_{\max}} \times \frac{T^{\text{self}}}{T}$). Project performance, as shown in the project performance row in Table II, is a result of outcome–output alignment, time spent on core activities, motivation score, and skill score. When the project performance equals 1, the project reaches its full potential. However, in reality, a project may not achieve the ideal performance due to correlations between the input variables. For example, communication and processes are necessary to coordinate between team members and align with company goals. Nevertheless, the performance scores for the four projects differ significantly and fall far short of the ideal state, suggesting room for improvement.

In short, OOL provides engineering teams and their management an easy way to understand their teams' strengths and weaknesses. Some are inherent in the nature of the projects, while others can be adjusted to achieve better outcomes. Applying the law on an ongoing basis rather than retrospectively would further help engineering teams celebrate success, identify areas for improvement, monitor progress, or justify changes.

V. DISCUSSIONS AND FUTURE WORK

This work presents evidence that engineering productivity follows OOL. It also illustrates the approaches to measuring engineering productivity based on OOL. The four projects are diverse and complicated real-world examples, but the sample size is relatively small to validate OOL fully. It is interesting to continue tracking the development of new product features in the company and apply these measurements to other companies. In Section III, we ignore second-order productivity contributions, such as software use cases, product development stages, software architectures, and programming languages. With more project data, we can explore how to include the second-order contributions through OOL analytically or numerically.

The work focuses on new product-feature development from project start to feature GA. However, before the start of projects, engineering, product, and senior management teams perform a large amount of work to test hypotheses, justify prioritization, and validate feasibility. The time to make decisions and allocate resources affects product-feature time to market, i.e., overall engineering productivity. We do not include this decision-making process in this study since it involves activities beyond the scope of software engineering, but this exclusion could be an interesting area from a new product development point of view.

Moreover, after feature GA, all project teams continue to develop their features in the form of new or enhanced additions to initial GA features, maintaining and bug-fixing existing features, fixing tech debt, or improving development infrastructure or processes. The continuing development is compounded to feature lifetime business success, which could be another area for future study.

OOL provides the connection between input metrics and output metrics. In this study, LOC and usage-based business impact are chosen as output and outcome metrics, and the percentage of time spent in activities, skill, and motivation scores are used as input metrics. However, OOL could be more universal than the metric combination examples we study in this work. Goals can be set for a project from financial, customer, internal operation, and employee learning and growth perspectives [17]. For example, story points, customer satisfaction scores, lifetime direct and indirect revenue of product features, or public benefits generated by nonprofits may be selected as output or outcome metrics, and one or a combination of DevOps Research and Assessment metrics may be used to represent infrastructure resistance [33]. Employee satisfaction scores, management ratings, and turnover rates may be used to reflect individual motivation or skills.

One purpose of the work is to formulate a quantitative OOL and test its validity. Therefore, we derive the quantitative OOL with measurable metrics from the original OOL variables, obtain the value of the metrics on both the left hand and right hand of (2), and test whether there is a linear relationship between the right-hand-side variables and the left-hand variables of the equation. For future applications, since the work supports the validity of OOL, we can make observations or obtain metrics for the left-hand side of the equation to understand or measure the right side of the equation, or vice-versa. We could also evaluate the engineering productivity impact of a management change by analyzing the change effect on one or several input metrics.

In this work, we evaluate OOL by employing historical product development projects. In the future, it is valuable to share OOL with engineers and managers, observe how OOL is used for productivity estimation or forecast, and document examples and findings for different management situations.

Additionally, the recent advances in generative artificial intelligence (AI) have sparked a wave of interest surrounding its potential impact on productivity, particularly within the realm of software engineering. Early studies show that generative AI improves engineering productivity, particularly for novice programmers [34], [35]. However, the overall implications of generative AI to real-world software product development are not clear. Can generative AI improve engineering team communication and collaboration or optimize process and infrastructure? Will it upscale engineer skills or motivation? Will it change the hypothetical limitation of engineering productivity? If so, how much is improvement? OOL may be an effective tool to shine a light on the productivity implications of emerging technology, including generative AI.

VI. SUMMARY

In this article, we explored how to quantitatively apply OOL to measure productivity in software engineering. We derived

practical and measurable metrics from the law. We discussed various approaches to obtain the measurable metrics. We demonstrated examples of using the law to measure industry product-feature development. The work supported the validity of OOL and showed how the law provides a comprehensive model to understand the engineering challenges of software feature development. We hope engineers and managers can apply OOL, metrics, and approaches in solving their own challenges.

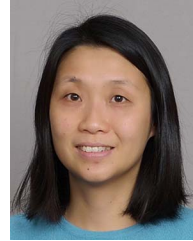
ACKNOWLEDGMENT

The authors would like to thank S. Shen, H. Kwok, D. Post, N. Hasija, P. Patnaik, D. Mathew, T. Britton, V. Jayanti, and A. Brown, who led the four studied projects, participated in the questionnaire and the lines of code extractions, and shared their engineering velocity observations, H. Hoffman for her diligent proofreading and insightful feedback, and the four anonymous reviewers for their valuable suggestions.

REFERENCES

- [1] M. E. Conway, "How do committees invent?," *Datamation Mag.*, vol. 14, no. 4, pp. 28–31, 1968. [Online]. Available: http://www.melconway.com/Home/Committees_Paper.html
- [2] F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*. Reading, MA, USA: Addison-Wesley, 1975.
- [3] Two-Pizza Teams. [Online]. Available: <https://docs.aws.amazon.com/whitepapers/latest/introduction-devops-aws/two-pizza-teams.html>
- [4] T. Godden, "Two-pizza teams are just the start, part 2: Accountability and empowerment are key to high-performing agile organizations," 2021. [Online]. Available: <https://aws.amazon.com/blogs/enterprise-strategy/two-pizza-teams-are-just-the-start-accountability-and-empowerment-are-key-to-high-performing-agile-organizations-part-2/>
- [5] N. Forsgren, J. Humble, and G. Kim, *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. Portland, OR, USA: IT Revolution Press, 2018.
- [6] N. Forsgren and M. Kersten, "DevOps metrics," *Commun. ACM*, vol. 61, no. 4, pp. 44–48, 2018. [Online]. Available: <https://dl.acm.org/doi/10.1145/3159169>
- [7] M. Skelton and M. Pais, *Team Topologies: Organizing Business and Technology Teams for Fast Flow*. Portland, OR, USA: IT Revolution Press, 2019.
- [8] N. Forsgren, M. Storey, C. Maddila, T. Zimmermann, B. Houck, and J. Butler, "The space of developer productivity," *ACM Queue*, vol. 19, no. 1, 2021. [Online]. Available: <https://queue.acm.org/detail.cfm?id=3454124>
- [9] D. A. Tamburri, F. Palomba, and R. Kazman, "Success and failure in software engineering: A followup systematic literature review," *IEEE Trans. Eng. Manage.*, vol. 68, no. 2, pp. 599–611, Apr. 2021. [Online]. Available: <https://ieeexplore.ieee.org/document/9104653>
- [10] M. Andreessen, "Why software is eating the world," 2011. [Online]. Available: <https://a16z.com/2011/08/20/why-software-is-eating-the-world/>
- [11] W. S. Humphrey and J. W. Over, *Leadership, Teamwork, and Trust: Building a Competitive Software Capability*. Reading, MA, USA: Addison-Wesley, 2010.
- [12] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby, "Cost models for future software life cycle processes: COCOMO 2.0," *Ann. Softw. Eng.*, vol. 1, pp. 57–94, 1995.
- [13] B. Boehm, C. Abts, B. Clark, and S. Devnani-Chulani, "COCOMO II model definition manual," Univ. Southern California, 1997.
- [14] A. J. Albrecht, "Measuring application development productivity," in *Proc. IBM Appl. Develop. Symp.*, 1979, pp. 83–92.
- [15] A. J. Albrecht and J. E. Gaffney, "Software function, source lines of code, and development effort prediction: A software science validation," *IEEE Trans. Softw. Eng.*, vol. SE-9, no. 6, pp. 639–648, Nov. 1983.
- [16] R. S. Kaplan and D. P. Norton, *The Balanced Scorecard—Measures That Drive Performance*. Boston, MA, USA: Harvard Business Review, 1992, pp. 71–79.
- [17] R. S. Kaplan and D. P. Norton, *The Balanced Scorecard: Translating Strategy Into Action*. Boston, MA, USA: Harvard Business Review, 1996.

- [18] M. Cohn, *Agile Estimating and Planning*. London, U.K.: Pearson Education, 2005.
- [19] H. Kniberg and A. Ivarsson, "Scaling agile @ Spotify with tribes, squads, chapters and guilds," 2012. [Online]. Available: <https://blog.crisp.se/wp-content/uploads/2012/11/SpotifyScaling.pdf>
- [20] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 33–53, Jan. 2007.
- [21] C. F. Kemerer, "An empirical validation of software cost estimation models," *Commun. ACM*, vol. 30, no. 5, pp. 416–429, 1987.
- [22] F. Liu and T. McKinnon, "Organizational Ohm's law—Case in point: Engineering velocity," *IEEE Trans. Eng. Manage.*, vol. 71, pp. 3832–3842, 2024.
- [23] M. Saunders, P. Lewis, and A. Thornhill, *Research Methods for Business Students*, 7th ed. London, U.K.: Pearson Education Limited, 2015. [Online]. Available: <https://www.amazon.com/Research-Methods-Business-Students-7th/dp/1292016620>
- [24] J. Křečková, D. M. Kennedy, H. Brožová, and J. Rydval, "Project management communication planning: An improved optimization model with additional recipients and individualized weights," *IEEE Trans. Eng. Manage.*, vol. 69, no. 4, pp. 1067–1080, Aug. 2022. [Online]. Available: <https://ieeexplore.ieee.org/document/9022932>
- [25] S. Krishan and M. Andreessen, "The observer effect: Marc Andreessen," 2020. [Online]. Available: <https://www.theobservereffect.org/marc.html>
- [26] State of Sales Report, 2023. [Online]. Available: https://www.salesforce.com/content/dam/web/en_us/www/documents/research/State%20of%20State%20-%202023.pdf
- [27] S. McConnell, *Software Estimation: Demystifying the Black Art*. Redmond, WA, USA: Microsoft Press, 2006.
- [28] K. Beck et al., "Manifesto for agile software development," 2001. [Online]. Available: <https://agilemanifesto.org/>
- [29] R. M. Parizi, P. Spoletini, and A. Singh, "Measuring team members' contributions in software engineering projects using Git-driven technology," in *Proc. IEEE Frontiers Educ. Conf.*, 2018, pp. 1–5.
- [30] J. Sutherland, A. Viktorov, J. Blount, and N. Puntikov, "Distributed Scrum: Agile project management with outsourced development teams," in *Proc. 40th Annu. Hawaii Int. Conf. Syst. Sci.*, 2007, Paper 274a.
- [31] M. E. Rogers, *Diffusion of Innovations*, 5th ed. New York, NY, USA: Free Press, 2003.
- [32] F. M. Bass, "A new product growth for model consumer durables," *Manage. Sci.*, vol. 15, no. 5, pp. 215–227, 1969.
- [33] N. Forsgren, M. C. Tremblay, D. VanderMeer, and J. Humble, "DORA platform: DevOps assessment and benchmarking," in *Proc. 12th Int. Conf. Des. Sci. Res. Inf. Syst. Technol.*, 2017, vol. 10243, pp. 436–440.
- [34] A. Ziegler et al., "Productivity assessment of neural code completion," in *Proc. 6th ACM SIGPLAN Int. Symp. Mach. Program.*, 2022, pp. 21–29.
- [35] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirel, "The impact of AI on developer productivity: Evidence from GitHub copilot," 2023, *arXiv:2302.06590*.



Fei Liu received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2002, the Ph.D. degree in electrical engineering from the University of California, Los Angeles, Los Angeles, CA, USA, in 2006, and the MBA degree in finance from New York University, New York, NY, USA, in 2011.

She is a Senior Emerging Tech Researcher with Okta, San Francisco, CA. She uses her technical and business skills to help Okta stay apprised of relevant market trends and technological developments. Prior to joining Okta, she held various research and strategy roles with Huawei and IBM. She is the author of more than 40 journal papers and conference presentations, and more than 70 patents. Over the years, her research interests have spanned from semiconductors, to identity and security, to organizational management.



Todd McKinnon received the B.S. degree in management and information systems from Brigham Young University, Provo, UT, USA, in 1993, and the M.S. degree in computer science from California Polytechnic State University, San Luis Obispo, CA, USA, in 1995.

He is the CEO and cofounder of Okta, San Francisco, CA, the leading independent identity provider, which he cofounded in 2009 with Frederic Kerrest. Under his leadership, Okta has transformed the way millions of people access technology and put identity at the forefront of security. His experience as a leader and technology visionary spans more than 25 years, with deep roots in enterprise software and cloud transformation. Prior to Okta, he led engineering with Salesforce.com, where he grew the team from 15 people to more than 250, and the service from 2 million daily transactions to more than 150 million. He started his career with PeopleSoft, where he held a number of engineering and leadership roles throughout his eight-year tenure.