Understanding Software Developer Activity via Windows 10 Telemetry

Spencer Buja, Tom Zimmermann, Philippe Kirsanov, Ankit Tandon,

Xue Liu, Jay Windsor, and Henok Addis

Microsoft

Wharton People Analytics Conference 2019

Abstract

Over the past several decades, businesses and the U.S. economy have become increasingly dependent on non-routine cognitive work. Business managers and researchers agree that it is difficult to measure and quantify productivity for this type of work, commonly referred to as “knowledge work”. Both startups and large enterprises must be able to analyze the productivity of their knowledge workers in order to thrive, as “knowledge work” becomes more prevalent. At Microsoft, most of the “knowledge work” is produced by software engineers that write code. Using novel data from the Windows 10 Operating System (OS), we have developed and tested hypotheses about how software engineers work and how to improve their experience. We demonstrate how this people analytics data is ethically collected, labeled, and analyzed to guide product and organizational decisions at Microsoft. Finally, we discuss the considerations and ethics of passively collecting productivity data.

Introduction

Non-routine cognitive jobs (i.e. “knowledge work”) require analytical, problem solving, and communication skills (e.g. managers, computer scientists, artists, architects, etc.) and account for 40% of U.S employment [1]. Such non-routine occupations (both cognitive and non-cognitive) have experienced an average of 2% growth every year [1]. Furthermore, routine occupations have showed significantly slower growth, less than 1% on average, or decline in the case of manual routine occupations [2]. The shift in demand for skills is commonly referred to as job polarization and is the result of outsourcing or automation of routine tasks. Since the process of job polarization is likely to continue, the picture is clear: a business that leads will be one that develops its workforce’s analytical skills and invests in tools and technologies that meet the demands of cognitive non-routine work.

One way to understand the productivity of knowledge workers is to track how they spend their time at work. Though time tracking tools exist for developers, their sparse utilization makes it difficult to draw meaningful conclusions. For example, in 2017 we found that less than 50 of the thousands of developers in Windows used an application called RescueTime [3]. In this paper, we introduce the usage of Windows 10 telemetry to automatically track and understand developer activity.

Windows 10 is uniquely positioned to apply these techniques, due to the breadth of applications that knowledge workers use in their jobs and the ease of IT deployment of Windows machines at companies. By writing simple database queries, organizations can obtain estimates of their developers’ time spent writing code. In the first part of this paper, we show how we compute the Code Authoring metric using Windows 10 telemetry. In the second part of this paper, we show how we used Windows 10 analytics to make an intervention to improve developer productivity.

Evaluating Time Spent Authoring Code within Microsoft with Windows 10 Analytics

Methods

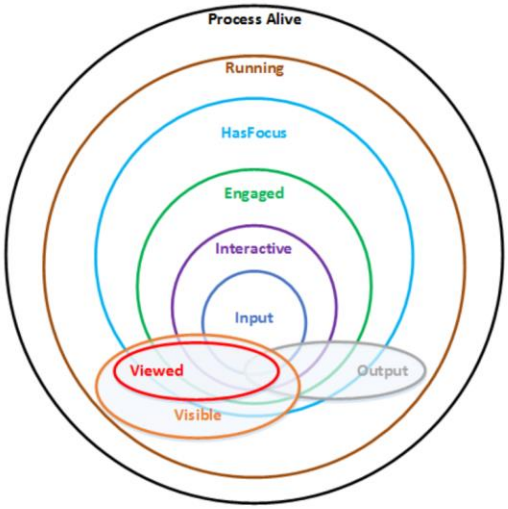
Overview In our study, we estimate the amount of time a software developer spends coding on their work computers in a time window (e.g. one day) using Windows 10 OS data. First, we compute a vector of sessions of activity data with the OS, which has limited context about the currently running application. Next, we generate labels, accounting for our uncertainty. Finally, we compute the dot product of the session vector with the label vector and return the Code Authoring metric in hours. We display the metric in a live information report. Figure I is a screenshot of the report.

Figure I. We report averaged daily Code Authoring times of software engineers by manager. In this case, we show the Code Authoring times of software engineers that report to Satya Nadella, Microsoft's CEO, with weekends removed. The large negative spike in the graph is caused by Labor Day in 2018.

Participants When computing the Code Authoring metric, we use the Windows 10 operating system activity analytics from the assigned work machines of groups of Microsoft employees in the software engineering discipline in the state of Washington. At this time, we do not calculate this metric for individuals within Microsoft, nor do we calculate this metric for users outside of Microsoft.

Procedures This study used a mix of database programming and business intelligence (BI) tools such as Cosmos, SQL Server, Kusto, and PowerBI. First, we made the choice about what type of Windows 10 OS telemetry to use. Figure II shows that application activity ranges from simply having a live process within the OS (most broad) to being immediately acted upon by user input (most specific). We chose to use the activity level of "interactive" because it best matched our intuition of development activity. Development activity requires some user input. However, measuring if there is input every second, the

definition of “Input” activity (see Figure II), is too high frequency.



Process Alive: The application has been launched and has a valid ProcessID ("PID") from the OS.

Engaged: The application is either Interactive (see following) or is in "DisplayActive" mode such that it has requested a keep-alive from the OS to prevent the system from entering sleep, turning off the display, and so on;

Interactive: The user has made some form of input (mouse, keyboard, touch, and so on) in the last 60 seconds.

Input: The user is making some form of input (mouse, keyboard, touch, and so on) in this moment (1 second intervals).

Figure II. On the left, we show a Venn diagram of the operating system activity levels available for analysis. On the right, we provide examples of descriptions of activity levels.

To compute a session with app interactivity, the telemetry system begins tracking if some form of user input has been applied to a Windows 10 application. It stops tracking if the employee begins providing user input to a different application or a 60 second timer runs out before there is any new user input to the application. To compute a vector of application times within a time window in our pipeline, we sum over application sessions on a single Windows 10 device. Then, we sum over each application in all Windows 10 devices assigned to Microsoft employees by querying the internal machine census database.

Next, we generated a set of labels for the applications running on Windows 10. We made a slight modification of the application labels from Table 6 in Meyer (et al.)’s[1] paper on Perceptions of Productivity [4]. Then, we created a tabular function to map applications to categories. Lastly, we created a second labeling scheme with richer semantics within the “Development: Other” and “Other” categories from the first set of labels. We labeled 373 applications using these two labeling schemes.

Table I. We used this first set of discrete labels to provide a general understanding of each application used by developers.

Label	**Description**
Development: Code	writing code
Development: Debug	debugging code
Development: Review	code reviewing
Development: VC	using version control

[1] We were unable to differentiate between development and non-development related activity on web browsers.

Development: TestApp	time spent testing
Development: Other	doing other relevant development activity
ReadWriteDocs	writing specs, plans, and documents
Email	reading and writing emails
Browse	browsing the internet
MeetInformal	messaging and meeting on communications applications
Irrelevant	interacting with lock screen, remotely accessing other environments, and using applications to share hardware resources between computers
Other	using entertainment applications
Uncategorized	using an unlabeled application

To create more expressive labels, we represented applications as mixtures of the labels in Table I. Because applications can be a mix of the labels in Table I, we stored probability mass functions for the discrete random variable of the label of the application. The original 373 labels were stored as deterministic distributions.

We collected 30 additional labels by following the typical crowdsourcing paradigm of *microtasking*. Here we break down the process of measuring how much time is spent coding into context-free units, the labeling of applications used on Windows 10 [5]. In an internal crowdsourcing tool, we collected Microsoft employee's beliefs about the categories of applications they were using. As with most crowdsourcing setups [5], we combined all labels together by taking the mean. We computed the mean of the discrete distributions of labels, which resulted in a discrete distribution. Taking a mean of crowdsourced data often results in less error. A single crowd worker can provide erroneous labels. For example, one individual labelled the Windows Photos app as a development tool. In the data collection tool, we experimented with modifying the metric using the intuition that employees on the same team use tools similarly. Then, we applied k-nearest neighbors to find using organizational tree similarity and averaged the ratings from k neighbors to model the belief of the label of an application.

Finally, with a vector of the degrees of belief that an application session is coding related and a vector of the same size of application times, we take an inner product of the two vectors to estimate time spent coding. We make this measure available for a given manager in an online PowerBI report that is internally available at Microsoft (see Figure I), providing the manager has a sufficient number of employees. In our reporting, we do not use the crowdsourced data.

Planning Developer Productivity Intervention using Windows 10 Analytics

Methods

Overview Another way we used Windows 10 telemetry to track developer activity is the product development of an internal tool called WAVE. WAVE is an IDE extension that enables Windows developers to use modern development environments driven by the hypothesis that a "golden path" solution for inner-loop workflows in the IDE will lead to less context switches and more satisfied developers. Before WAVE, Windows developers were not able to leverage modern IDEs due to the size

of the Windows code base, a custom build system, and the lack of tools that make sense of the Windows build graph without executing a build.

While planning the product development, the WAVE team asked what the most common editing tools are at Microsoft to decide what development environments to support. Using data from the Windows telemetry platform, the WAVE team decided to support VS and VS Code. Finally, as the tool has rolled out, the team monitors Windows 10 analytics.

Participants When computing the Code Authoring metric, we use the Windows 10 OS analytics from the assigned work machines of Microsoft employees in the software engineering discipline in the state of Washington. The software developers contributing to the Windows OS is a subset of the potential population that can be measured.

Procedures Between November 1, 2017 and May 1, 2018, we collected participants' engagement hours and filtered it to contain all sessions in the "Code Editing" and "IDE" categories. These categories are elements of the second version of application categories and do not appear in Table I.

With the engagement data, we calculated two metrics. First, we calculated the distinct number of users of each of these applications. Second, we calculated the total engagement time, summed over all participants. We computed these two metrics for both the six-month window and the last three months of the window. This data is available in Table II, sorted by distinct counts of employees that used the program at least once in the six-month window. Visual Studio and Visual Studio Code are ranked very highly in Table II.

Table II. We estimated the top 15 integrated development environments and editors at Microsoft using counts of distinct employees via Windows 10 telemetry in a six-month window.

Application Name	Distinct User Count (Employees)	Total Engagement (Hours)
Notepad	16,776	34,642.22
Visual Studio	13,745	443,572.88
Notepad++	7,444	31,696.15
Visual Studio Code	4,993	64,555.36
Nodepad2	1,743	1,649.46
Sublime Text	1,623	11,748.87
Visual Studio Enterprise 2015 with Updates	879	51.30
Unity.exe	511	3,125.44
Unity Editor	488	2,568.48
Android Studio	458	4,339.17
Microsoft Visual Studio Enterprise 2015	451	32.83

VI Improved (VIM)	422	3,834.99
Source Insight Program Editor	384	6,558.88
Atom	216	304.32
Source Insight 4.0	183	1,779.15

Over the next few months, the WAVE team developed a version of WAVE for Visual Studio and a version of WAVE for Visual Studio Code. Next, we limited the participants to Windows Core OS developers and created a PowerBI report to view the trend of code editor and development environment usage over time.

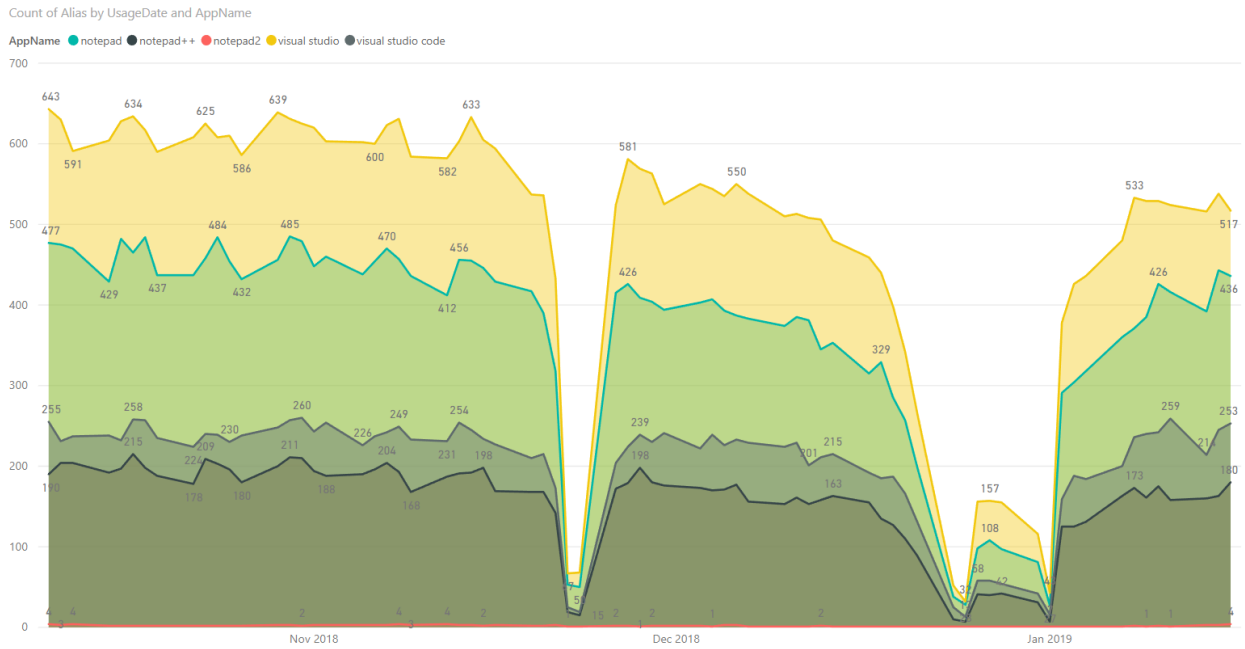


Figure III. This chart shows Windows 10 telemetry for the COSINE (Core OS) organization in Microsoft.

Discussion

Careful and targeted evangelization using the Windows 10 telemetry has led to great successes in Monthly Active Users for WAVE, jumping from ~300 users in November to 739 in February.

Limitations

The two studies in this white paper have limitations. First, an assumption of the Code Authoring metric is that software developers are typing in an application or IDE when writing their code. This may not be true, as files can be edited with the online editors in websites that host code repositories and version control such as GitHub and Azure DevOps. However, based on the experience of the team, the number of people using online code editors appears to be insignificant. Additionally, respondents to the Stack Overflow 2018 developer survey do not mention using a browser or online editor as the primary development environment of the individual developer [6].

Second, another requirement of the Code Authoring metric is that we have labels for the Windows 10 applications used by employees. We have not labelled every application used on the Windows 10 platform and new applications arrive frequently.

Key Takeaways and Future Directions

We are excited about using Windows 10 to analyze the activity of software engineers and, more broadly, knowledge workers. Internally at Microsoft, Windows 10 analytics has been used to both discover new opportunities and to create new evaluative metrics to understand interventions' impact on developer productivity. In the future, other businesses will be able to use Windows 10 to ethically monitor and improve the productivity of their knowledge workers.

The most important topic for the future direction of this research is the ethics of the use of algorithms and analytics to compute productivity data about knowledge workers. Newman, Fast, and Harmon investigated the ethical theories that help us draw ethical conclusions about the use of analytics for automated decision-making [7]. Utilitarian ethics proclaims we should maximize societal outcomes efficiently and distribute outcomes fairly (e.g. based on productivity), which indicates that we should investigate productivity analytics. However, in Kantian ethics, we should never treat humans as means to ends. Therefore, we should not reduce humans to numbers. The conclusion of Newman (et al.) is that we should utilize both human judgment and analytics to make decisions. Similarly, Dietvorst and Massey (et al.) found that humans are more comfortable with using algorithmic decision-making systems based on data modeling, if outputs are modifiable by their human users [8]. Thus, we strongly believe that good human judgment must be used when drawing conclusions with coding time estimates and other developer productivity metrics.

Second, we need to understand how Windows 10 analytics relates to the two standard ways of measuring developer productivity. Though there is no generally accepted quantitative measure of productivity, researchers have measured it by quantifying how much the engineer has affected their product. Traditionally, researchers have used lines of code [9] and function points [10]. However, developers have presented valid arguments that they can deliver value by deleting dead source code and making one-line code changes for critical bug fixes. Other researchers analyze self-reported productivity by developers [11]. It is crucial to understand how time spent coding relates to both traditional measures of productivity and self-reported productivity. This will help verify the utility of organizations striving to improve their Code Authoring time.

Furthermore, the WAVE extension for OS developers is still under active development. By leveraging the Windows 10 telemetry, we have begun testing our hypotheses about the code authoring experience for developers in Windows and whether context switches are reduced for WAVE users. This telemetry has also allowed us to directly engage with developers on non-WAVE toolsets to collect feedback on old workflows and evangelize the WAVE workflow.

Finally, we must address the impact of browsers. The top two applications used by the participants are internet browsers. The activity on the internet browser is as diverse as an operating system because they are both platforms for applications. Understanding and categorizing browsers will dramatically improve the quality of Windows 10 analytics.

References

[1] M. Dvorkin, “Jobs Involving Routine Tasks Aren’t Growing,” 2016. [Online]. Available: https://www.stlouisfed.org/on-the-economy/2016/january/jobs-involving-routine-tasks-arent-growing .

[2] M. Dvorkin, “The Growing Skill Divide in the U.S. Labor Market,” 2017.

[3] “RescueTime.” [Online]. Available: https://www.rescuetime.com/. [Accessed: 02-Apr-2019].

[4] T. Zimmermann, A. N. Meyer, G. C. Murphy, and T. Fritz, “Software Developers’ Perceptions of Productivity,” *ACM - Assoc. Comput. Mach.*, 2014.

[5] Walter S. Lasecki, “Hybrid Intelligence Crowdsourcing for Robust Interactive Intelligent Systems.”

[6] “Developer Survey Results 2018,” 2018. [Online]. Available: https://insights.stackoverflow.com/survey/2018.

[7] D. Newman, N. Fast, and D. Harmon, “When Eliminating Bias Isn’t Fair: Analytics, Algorithms, and Procedural Justice.” [Online]. Available: https://vimeo.com/257794509.

[8] B. Dietvorst, J. P. Simmons, and C. Massey, “Overcoming Algorithm Aversion: People Will Use Imperfect Algorithms If They Can (Even Slightly) Modify Them,” 2015.

[9] Frederick P. Brooks, *The Mythical Man-Man*. 1972.

[10] A. J. Albrecht, “Measuring Application Development Productivity,” *Proc. IBM Appl. Dev. Symp*, pp. 83–92, 1979.

[11] 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 Trans. Softw. Eng.*, pp. 1178–1193, 2017.