Workgraph: Personal Focus vs. Interruption for Engineers at Meta

Yifen Chen, Peter C. Rigby, Yulin Chen, Kun Jiang, Nader Dehghani, Qianying Huang, Peter Cottle, Clayton Andrews, Noah Lee, Nachiappan Nagappan
Meta Platforms, Inc.
Menlo Park, New York, Seattle, and Bellevue, USA
yifenchen,pcr,yulinchen,kunjiang,naderld,qyhuang,pcottle,clayton,noahlee,nnachi@fb.com

ABSTRACT

All engineers dislike interruptions because it takes away from the deep focus time needed to write complex code. Our goal is to reduce unnecessary interruptions at Meta. We first describe our Workgraph platform that logs how engineers use our internal work tools at Meta. Using these anonymized logs, we create personal-focus sessions. Personal-focus sessions are defined in opposition to interruption and are the amount of time until the engineer is interrupted by, for example, a work chat message.

We describe descriptive statistics related to how long engineers are able to focus. We find that at Meta, Engineers have a total of 14.25 hours of personal-focus time per week. These numbers are comparable with those reported by other software firms.

We then create a Random Forest model to understand which factors influence the median daily personal-focus time. We find that the more time an engineer spends in the IDE the longer their focus. We also find that the more central an engineer is in the social work network, the shorter their personal-focus time. Other factors such as role and domain/pillar have little impact on personal-focus at Meta.

To help engineers achieve longer blocks of personal-focus and help them stay in flow, Meta developed the `auto-focus` tool that blocks work chat notifications when a engineer is working on code for 12 minutes or longer. `auto-focus` allows the sender to still force a work chat message using "@notify" ensuring that urgent messages still get through, but allowing the sender to reflect on the importance of the message. In a large experiment, we find that `auto-focus` increases the amount of personal-focus time by 20.27%, and it has now been rolled out widely at Meta.

ACM Reference Format:
Yifen Chen, Peter C. Rigby, Yulin Chen, Kun Jiang, Nader Dehghani, Qianying Huang, Peter Cottle, Clayton Andrews, Noah Lee, Nachiappan Nagappan. 2022. Workgraph: Personal Focus vs. Interruption for Engineers at Meta. In *Proceedings of The 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

[*] Rigby is an associate professor at Concordia University in Montreal, QC, Canada.

ESEC/FSE 2022, 14 - 18 November, 2022, Singapore

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Software engineering is a complex and intellectually demanding job. At Meta, we have heavily invested in ensuring that our engineers attain work-life balance. For example, our balance bot allows employees to specify their working hours and blocks messages outside these hours. Continuing the effort at Meta to make engineers productive and happy, we focus on interruptions. Prior work in industry have found that self reported "good days" have fewer interruptions [7], that interruptions are a serious problem at their company [6], and that interruptions negatively disrupted task performance, cognitive load, and put users in a negative emotional state [1]. Although most engineers can recover from an interruption in around 15 minutes [8], when working on complex tasks interruptions are more difficult to recover from because of an increase cognitive load [3].

In this work, we first provide descriptive statistics on the level of undisrupted personal-focus that our engineers achieve. We create a model to understand the factors that lead to longer focus. Finally, we continue to enhance engineers worklife balance by automatically blocking Workchat (Meta's internal messaging platform) notification when an engineer has been working on code for 12 or more minutes. This work contributes to basic research into how long people remain focused by using fine-grained logged data. We discuss how we collect this data. We address the following research questions.

RQ 1. Focus vs. Interruption: How long do engineers remain focused?

Prior work has used surveys (*e.g.,* [7]) or IDE logged events (*e.g.,* [8]). In contrast, we collect information on the full range of tools that engineers at Meta use giving us a clear view of when they are able to focus.

Findings. We describe the Workgraph platform that we developed to help improve the experience of software engineers at Meta. When applied to personal-focus we find that the sum of time spent in personal-focus has a median 19.29 hours/week. When we restrict this time to flow sessions that are more than 12 minutes, we see a median of 14.25 hours/week.

RQ 2. Focus Factors: Which factors affect the length of daily personal-focus work sessions?

We want to understand the factors that affect how long engineers remain focused. We perform a Random Forest regression on the median daily personal-focus and use engineer tenure, centrality, work pillar, and VSCode actions to understand factors that affect focus.

Findings. Median daily personal-focus for engineers is increased when they perform more actions in IDE editors, with a feature importance of 0.48. The work related social network centrality of

an engineer reduces their focus time (importance 0.37) as they coordinate more work. The remaining factors such as tenure, role, and pillar have little explanatory power with each having a feature importance less than 0.06, implying that personal-focus is consistent regardless of pillar/domain and engineering role.

RQ 3. `auto-focus`: When an engineer is focusing, what is the impact of automatically blocking interruptions?

In a remote work setting, the in-office queues of a door closed or headphones on are not present to indicate to others that an engineer is in deep focus. Meta developed a tool that uses our Workgraph platform to block Workchat messages when an engineer is coding for 12 or more minutes. The message sender is notified that their message was blocked and has the option to override the block, '@notify' for urgent messages that need a synchronous response.

We conduct an experiment to determine if blocking these notifications increases personal-focus time. We use a guard measure[1] to ensure that the overall number of messages does not decrease as we still need engineers to communicate and collaborate.

Findings. In a large experiment of `auto-focus`, we found that after engineers started using `auto-focus` to block interruptions after 12 minutes of coding, their median personal-focus time increased by 20.27%. We do not see a decrease in the number of messages sent. `auto-focus` has been rolled out widely at Meta.

This paper is structured as follows. In Section 2, we provide background on our internal tooling and Workgraph platform and define personal-focus sessions. In Section 3, we provide descriptive statistics on how long engineers are able to focus at Meta. In Section 4, we examine how the centrality of an engineer and other factors influence personal-focus time. In Section 5, we describe `auto-focus` and how a simple tool has improved engineer personal-focus. In Section 6, we discuss threats to validity. In Section 7, we position our findings in the larger related work. In Section 8, we conclude the paper.

2 BACKGROUND ON THE WORKGRAPH PLATFORM

At Meta thousands of software engineers rely on and build upon our internal infrastructure to build social value and economic opportunity for our users and businesses everyday. The software stack of our infrastructure comprises a rich repertoire of internal tools, frameworks, and platforms covering different domains such as system, data, code, and ML/AI authoring and management. Beyond the technical domains we also build our own internal work, time, focus, and collaboration management products to help our teams perform their work. Most of our stack is developed in-house, which provides us with a unique opportunity for consolidating and standardizing the telemetry of our infrastructure to make data-driven decisions, optimize our software engineering processes, and build experiences that our engineers love.

2.1 Workgraph Architecture

Figure 1 shows the architecture. Workgraph is a meta-platform with a data, analysis, front-end/back-end layer and various insight solutions built on top of the platform. The "meta" prefix indicates that each layer in itself follows a platform model with the aim to optimize network of network effects and that the co-existence and integration of other internal platforms is possible. The platform consists of three layers that tightly integrate with our technical ecosystem. In this study we will primarily focus and explain how the data layer works.

The data layer - This layer comprises an event driven data model we call Data Confidence. Tools and products span data, developer, ML/AI, productivity, and various other domains. The data consists of a hierarchical data model that models tool specific logging events at the raw, daily, and multi-day aggregation level. The hierarchy allows us to link high level patterns or metrics with the raw event data.

An important design element of our data model is the event. Interactions between the tools and users are logged as events. Each event has a rich context associated with it such as the user, timestamp, application type, event type, feature, duration, error, state, resources, and meta information. The application, event, and feature types provide us a holistic understanding on engineering behavior, i.e. what tools, features, and activities are being performed when engineers interact with our tools. The duration information provides us with context around the performance of the tools (e.g. waiting times) and the error a notion around user-facing reliability (e.g. task success rate). To provide an example a developer clicks on a button to compile their code in VSCode, but after 10 minutes the event fails with an error, which prompts the user to trigger a chat message to a co-worker to ask for help. The state information captures the code and build information such as the relevant files, libraries, and build targets. The resource information captures the associated resource identifies such as the pull request ID. Behavior, performance, reliability, and the associated state and resource context are key aspects to understand engineering productivity as well as the challenges.

One of the key challenges to address was the problem of distinguishing between human triggered and system triggered events to properly estimate human vs. machine time spent. As our internal logging frameworks do not enforce this difference we employ a set of heuristics and ML models to label human vs. machine triggered events in our data model. For example, we filter out events that happen at regular exact intervals as such events very unlikely can be performed by engineers. We also exploit the fact that engineers usually work normal hours and filter out events during non-work time vs. normal and oncall work times. We also leverage ML models to flag if a certain event type and feature is a human vs. machine triggered event (e.g. ClickButtonAction vs. BatchSyncUpdate). The model predictions and model interpretability analyses, such as feature importance and error cases, are used by human reviewers to validate the flags and continuously monitor and update our data model and chosen heuristics.

We performed a variety of optimizations to ensure consistency and convenience in our data model for upstream use such as naming conventions, timezone mapping, retention periods, and logging semantics. Due to the inconsistency of how tools and products perform their logging we needed to consolidate naming schemes (e.g. FBLearner vs. FBLearn) and conventions as well as disambiguate name and version changes (e.g. Tool_V1 vs. Tool_V2). For consistency we also enforce that application names are lower case, multiple words are separated with an underscore, and the absence

[1] Guard measures are measures that should not be impacted by a change.

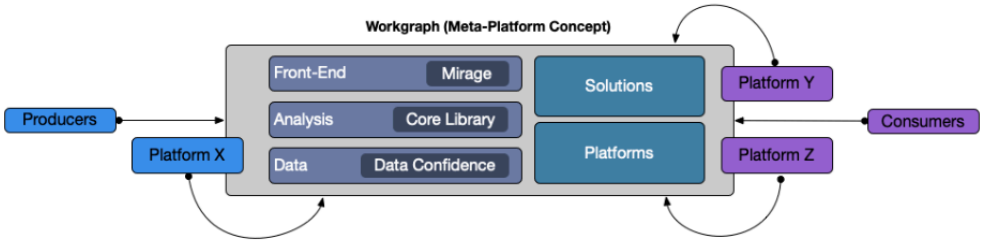

Figure 1: An overview of the high level architecture of the Workgraph meta-platform. The platform provides the foundations and interface for Producers and Consumers to provide and consume value. At Meta we have various platform offerings, which can serve as producers and/or consumers. The data, analysis and front-end layer of Workgraph each follows a platform concept. We also support the notion that sub platforms can coexist within Workgraph. The architecture is highly interconnected. The Workgraph core layers and co-hosted platforms are leveraged to build solutions.

of special characters. To account for our global workforce and support time-zone specific analysis we provided convenience functions to map between different time zones. Every tooling source table has their own specific retention period, which can range from 1 to 3 days to multiple years. We implemented data pipelines to ingest the source data from various tools and products at a daily cadence to build up a longer term history of our data model for longitudinal analysis. Lastly, every tooling source table can have their own logging semantics such as capturing data on a sequential or cumulative basis. Column names also vary as well as their values (e.g. seconds vs. milliseconds). Due to the scale of our infrastructure it was not possible to analyze each table separately. So instead the team prioritized the top-50 tools to perform detailed manual analysis of the tooling source tables to ingest the data into our data model. The primary criteria was to include tools based on the biggest user volume and time spent. We also included tools that are used less, but are considered important as part of the developer workflow such as service outage tooling or debugging tools. For the long-tail we created a self-service "runbook" with design principles and guidance so other teams can ingest their data in a consistent manner.

The analysis layer - This layer comprises a Python library we call Core Library, which serves as a unifying wrapper library for our internal APIs spanning system, data, code, and ML/AI frameworks. It also consists of custom algorithms and convenience functions used by our engineers to i) load, transform, and create data, ii) perform analysis, and iii) build ML models.

The front-end layer - This layer comprises two components. Insight solutions that are built on top of the existing SaaS infrastructure and a no/low-code visual programming interface to provide easy access to Data Confidence, Polymath, Core Library, and our internal APIs.

Workgraph is used by hundreds of teams throughout the company empowering teams to make the right decisions faster and empower building experiences that our engineers love.

2.2 Defining personal-focus Sessions and Interruptions

Engineering is often collaborative, but requires blocks of personal-focus to complete complex tasks. In this work, we define personal-focus time in opposition to interruptions. Interruptions are defined as using Workchat (sending a message/reaction), using the calendar, unscheduled calls, interacting in Workplace[2], and clicking a notification. We define personal-focus to be active time spent on a work computer without interruptions. We use a threshold to define the minimum length of a session because existing research evidence states that a developer takes between 10 to 15 minutes to get back into flow state [8]. To avoid overcounting when an engineer walks away from their computer, we end a session when there is no activity for 30 minutes. Instead of using the entire 30 minutes for this session, we use the median length of time the tool is used across all engineers.

An example of personal-focus sessions and interrupted sessions are shown in Figure 2. In this example, we use 15 minutes as the minimum length of a personal-focus session. The engineer uses several personal-focus tools (VSCode, search, and wiki) for 15 minutes, and then starts a conversation in Workchat for a minute (to get some wiki link from a coworker). After checking the wiki page, this employee then started to check several workplace pages and send messages for another 5 minutes. In this example, we have 4 sessions and only the first and last are considered non-distracted sessions. The third session with one Wiki page viewing activity would be non-distracted, but it only lasts for 1 minute, which does not meet the threshold. To study personal-focus we use the logged Workgraph data between June 15, 2021 and December 15, 2021. The methodology used to provide answers for each of our research questions differs, so we describe the method with each research question.

3 RQ 1. FOCUS VS. INTERRUPTION

How long do engineers remain focused?

For this research question, we provide descriptive statistics regarding the weekly sum of personal-focus for the median engineer.

[2]Workplace functions like Facebook, but is for sharing work notes and connecting with colleagues www.workplace.com

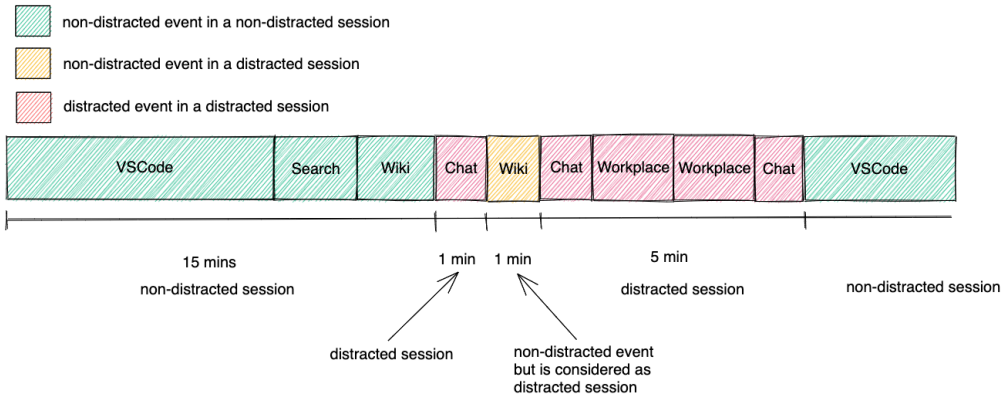

Figure 2: A personal-focus session is defined as a non-distracted session that lasts for 15 minutes or more

Table 1: The median developer's sum of personal-focus time and number of sessions per week

Threshold	Focus time	Sessions
> 0 minutes	19.29 hr/week	143
≥ 3 minutes	18.08 hr/week	56
≥ 12 minutes	14.25 hr/week	27
≥ 15 minutes	13.45 hr/week	23
≥ 30 minutes	9.67 hr/week	10
≥ 60 minutes	4.80 hr/week	3
≥ 90 minutes	3.82 hr/week	2

We report the length and number of each personal-focus session in Table 1 varying the threshold for the minimum length of a personal-focus session.

We can see that the sum of total time spent in personal-focus has a median across all engineers at Meta is 19.3 hours per week and 143 focus sessions. Many of these sessions are micro-sessions because when we examine personal-focus sessions with a minimum of 3 minutes, we see that there is a 61% decrease in the number of sessions (56). However, there is only one hour less personal-focus time per week indicating that these are much longer focus sessions. This trend continues as we increase the minimum length of a personal-focus session. Software engineering is an inherently collaborative activity and we see that engineers need many micro-sessions to ask questions from others to get their work done.

In contrast to these micro-sessions, studies show that longer focus blocks of uninterrupted time are required to attain flow. While the exact minimum time necessary for flow is controversial, most studies agree that flow begins at around 15 minutes [3, 8]. At Meta, when we set the threshold at 15 minutes, we see that in the median case, engineers have 13.45 hours/week of flow time and 23 sessions per week. Engineers also have much longer flow sessions, with a median of 2 sessions at 90 minutes or longer resulting in 3.82 hours per week of very long personal-focus.

We find that the sum of time spent in personal-focus has a median 19.29 hours/week. When we restrict this time to flow sessions that are more than 12 minutes, we see a median of 14.25 hours/week. We also see that engineers have a median of 3 sessions that last longer than 1 hour, with those sessions lasting a median of 4.80 hours/week.

4 RQ 2. FOCUS FACTORS

Which factors affect the length of daily personal-focus work sessions?

After quantifying how long engineers can stay focused, we next ask what other factors contribute to the length of personal focus sessions. We model each developers' median daily personal-focus time using a Random Forest model based on the following features: VSCode usage, centrality of the engineer, tenure, engineering role, and organizational pillar.

4.1 RQ2. Features and Model

The features used in the model are shown in Table 2 and described here.

Tenure: we suspect that the the number of years an individual has been at Meta will impact how long they are able to focus on their work. New engineers will likely need to ask many questions from senior colleagues.

Centrality: we included the centrality of the engineer based on the degree centrality of each engineer from Meta's internal employee social network graph, where the interactions include meetings, chat messages, document collaboration, and code review interactions.

Engineering role: since different roles may require different responsibilities and working patterns, engineering role is included in the model as we suspect it may impact personal-focus time. In the model, we categorize this feature into Software engineering (SWE), which is one of the dominant roles at Meta, and Other engineering.

Organizational pillar/domain/product: there are many pillar/domains across Meta with different focus areas, which may directly affect how engineers focus and communicate in their day-to-day duties. We picked a few of the pillars with largest correlation with personal-focus time and grouped the rest into a single category.

Table 2: Description of the features that we use in our model of median daily personal-focus time.

Feature	Description
Tenure	The number of years that an engineer has been at Meta
Centrality	Number of connections in employee social graph
Engineering Role	A categorical variable divided by Software Engineering and Other Engineering (including Production Engineering)
Pillar	A categorical variable describing the pillar or suborganization. Examples include Facebook App, Infrastructure, etc
IDE actions	A count of the number of actions an engineer makes in primary editors used at Meta: VSCode, Android Studio, and Xcode.

Table 3: The permutation importance of features influencing median daily personal-focus time. We replace the actual pillar name with a letter to preserve confidentiality.

Feature	Importance	Direction
1. IDE Actions	0.482	↑
2. Centrality	0.374	↓
3. Tenure	0.063	↓
4. isSoftwareEngineer	0.017	↑
5. Pillar A	0.020	↓
6. Pillar B	0.015	↑
7. Pillar C	0.015	↑
8. Pillar D	0.013	↓

IDE actions: we count the number of actions an engineer makes in the integrated development environment (IDE) editors, including VSCode, AndroidStudio, and XCode. For engineers, we expect that more activity in the editors will likely correspond less time spent in meeting and other non-focused work.

We built a *Random Forest* regression model using the features described in Table 2. Our model was trained on the median of the personal-focus sessions among 23K engineers and tested on the median of the personal-focus sessions among 5.8K engineers between June 15, 2021 and December 15, 2021 (6 months). We evaluate model fit using the adjusted R^2. To evaluate the importance of each feature, we use the feature importance package from scikit-learn.[3] This determines the decrease in the score when each feature is removed and indicates its importance.

The final model has the following form:

MedianPersonalFocus ~

Centrality + Tenure + IDE_Actions + isSoftwareEngineer + Pillar

4.2 RQ2. Results

The Random Forest regression model of median daily focus time for engineers has a training adjusted $R^2 = 0.88$ and testing adjusted $R^2 = 0.31$. The permutation feature importance and impact on the direction of personal-focus are shown in Table 3. We discuss each in order of importance. (1) we see that IDE usage is the main contributor to the personal-focus time among engineers with an importance of 0.48. Clearly the longer an engineer spends in the IDE, the more time they have to work on coding tasks that require deep focus. (2) The centrality of an engineer has an importance of 0.37. The more central an individual is, the less time they have to spend on tasks that need personal-focus because they are assisting other individuals and coordinating work. The remaining features have minimal importance to the model. 3. Tenure, the longer an engineer is at Meta the shorter personal-focus time they have. Tenure has a Spearman correlation of 0.59 with centrality as more senior engineers tend to be more central. 4. Software engineers have slightly more focus time than other types of engineers. The pillar/domain has negligible importance with engineers in all domains demonstrating similar amount of median personal-focus.

> Median daily personal-focus for engineers is increased when they perform more actions in IDE editors, with a feature importance of 0.48. The centrality of an engineer reduces their focus time (importance 0.37) as they coordinate more work. The remaining factors such as tenure, role, and pillar have little explanatory power with each having a feature importance less than 0.06, implying that personal-focus is consistent regardless of pillar/domain and engineering role.

5 RQ 3. AUTO-FOCUS

When an engineer is focusing, what is the impact of automatically blocking interruptions?

Our first attempt to improve engineers' focus was to allow them to explicitly create "Focus Blocks" in the calendar. During these times, messages would not be push notified and engineers would not be interrupted. Unfortunately, creating and maintaining these blocks meant that usage was low and we did not see an improvement in focus time [2]. Instead, at Meta we know when an engineer is focusing on work vs. collaborating with colleagues over chat or in meetings. The goal of `auto-focus` is to *automatically* block chat notifications when a developer is in deep coding focus. For `auto-focus` we define deep focus to be working on a coding tasks for 12 or more minutes. When the engineer receives chat messages during deep focus, the sender will receive the following with an example displayed in Figure 3: "This message did not push-notify to <Engineer's name> because they are currently in deep focus while

[3] https://scikit-learn.org/stable/modules/permutation_importance.html

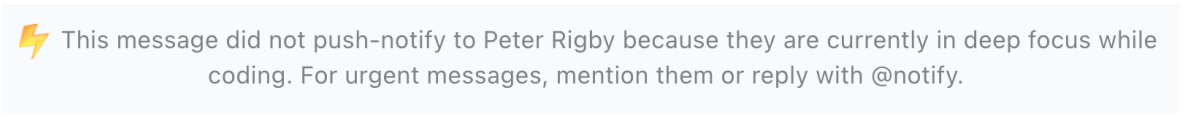

Figure 3: When a Workchat message is sent and an engineer is focusing on code, the sender will be notified that their message was not pushed notified to the engineer. The sender can then reflect if the message needs to be answered immediately "@notify" or if it can wait until the engineer has a natural break in focus.

coding. For urgent messages, mention them or reply with @notify." If the sender has important information, such as a response to a coding question, or something urgent, *e.g.*, a crash, they can use "@notify" and `auto-focus` will allow the message to interrupt the engineer. As a fictitious but representative example of `auto-focus` in action. The generic "Hi Peter" message is blocked and Nachi is notified that Peter is in deep focus coding. However, Nachi has important information for Peter and he responds, "@notify system X has seen a performance drop, we need to start debugging this issue right now, let's meet." This message is pushed to Peter by-passing `auto-focus` and allowing them to work immediately on the issue. The design of `auto-focus` allows the sender to reflect on the urgency of their message and to consciously and explicitly decide when to interrupt a colleague.

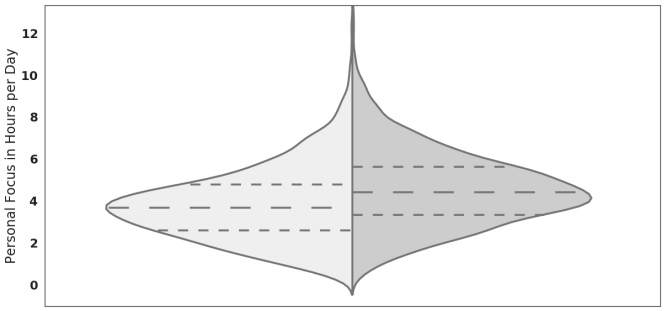

Figure 4: Left violin shows the personal-focus time per day before `auto-focus`. The right shows personal-focus time with `auto-focus`. We see a median increase of 20.27%.

5.1 RQ3. Experimental Design

All significant changes at Meta go through A/B experimentation. In an A/B trial, the old feature 'A' is experimentally compared with the new feature 'B.' In our case, the control group A does not have any Workchat notifications blocked by `auto-focus`. In the test group B, `auto-focus` blocks chats giving the author the chance to force a notification, "@notify."

Our goal is to reduce the interruptions and increase focus, but we do not want to inadvertently decrease the level of communication among engineers. Each experiment at Meta has goal metrics and guard metrics. The goal metric in this experiment is the length of median daily personal-focus time. We hypothesize that `auto-focus` will increase median personal-focus time. The guard metric ensures that blocking notifications does not have an adverse impact. For `auto-focus` our guard measure is the daily number of chat messages sent. We expect to see no statistically significant change in the number of message sent. The ideal outcome of this study is that the same number of messages are sent, but that the messages are responded to when the engineer is not in deep focus and are taking a natural break.

In this experiment, we study 3.3k engineers who were enrolled into `auto-focus` between July 2021 and March 2022. Instead of using a traditional A/B test, we compare each engineers before and after `auto-focus`. We exclude the winter holiday and the performance evaluation cycle at the start of the new year from the analysis because engineering work patterns are systematically different in these periods. To compare the goal and guard metrics before and after `auto-focus` use, we use a Wilcoxon test because the data is not normally distributed. We use two sided violin plots to display the metric distributions. A violin plot is a kernel density plot of the distribution. The left of the violin plot shows the metric before `auto-focus` and the right shows `auto-focus` usage, we also show the 25th, 50th, and 75th percentiles, i.e. P25, P50, and P75.

5.2 RQ3. Results

The daily personal-focus time distribution is shown in Figure 4. When we compare the before `auto-focus` we see a median daily personal-focus of 3.69 hours compared to a median of 4.43 when using `auto-focus`. This represents an increase of 20.27% in daily personal-focus when using `auto-focus`. Comparing these distributions with a Wilcoxon test, we see that the results are statistically significant with a $p \ll 0.001$.

Our guard metric of the number of messages sent surprisingly increased from a median daily messages of 18 to 21. Comparing the distributions with a Wilcoxon test we have $p \ll 0.001$. Although future work is necessary to understand this increase, we conjecture that it may be because engineers are focused on responding to messages rather than splitting their attention between Workchat and trying to write code. In effect, it may be that because the engineer has been able to focus on code, they are now more able to focus on providing a focused response to a colleague's messages.

> In a large experiment of `auto-focus`, we found that after engineers started using `auto-focus` to block interruptions after 12 minutes of coding, their median personal-focus time increased by 20.27%. We do not see a decrease in the number of messages sent. `auto-focus` has been rolled out to all employees at Meta.

6 THREATS TO VALIDITY

In Section 2, we discussed the Workgraph platform. It is impossible to validate all logged data, however, we have worked with engineers on other teams to validate their logged sessions. We also use the median time of the final tool usage to avoid counting time when an engineer has walked away from their work computer. The descriptive statistics in Section 3 are consistent with values reported in other works, *e.g.,* [8]. This type of replication work in new contexts is an important part of generalization. However, it is unclear whether our results will generalize beyond Meta to other development contexts. With remote work increasing and the loss of in-person "busy" signals, we hope that other organizations will build upon `auto-focus` to adapt it to their context.

We did not perform any hyperparameter tuning for our model of personal-focus time, which leaves open the possibility that these models can be further improved through a rigorous hyperparameter optimization. By using unoptimized defaults, the hyperparameters were fixed before looking at the test set to ensure no contamination. In addition, there is no reason that a hyperparameter search on Random Forest would result in dramatic changes in which features are considered the most important, which is what we focused on in the end. The first hyperparameter of 'depth' would mostly impact features that are less important, since most major features would already be accounted for in the first few layers of the model's trees. In addition, the other hyperparameter of 'number of trees' would be expected to have negligible impact on feature importances given that each tree in the Random Forest is independent. We used the median personal-focus time, however, it is possible that other values of personal-focus time could lead to different results. We conducted a basic sensitivity analysis and in addition to the median of personal-focus sessions, we also ran the models on P50 and P75 of personal-focus sessions and obtained similar training adjusted R^2 (0.89 and 0.88 respectively) and testing adjusted R^2 (0.34 and 0.30 respectively). We do not expect other modelling methodologies or thresholds to substantively change our results.

7 RELATED WORK

There has been substantial work in understanding interruptions. Surveying over 5k developers at Microsoft, Meyer *et al.* [7] found that good workdays had statistically significant fewer interruptions than typical or bad workdays. Furthermore, self-reports from developers indicated an average of 47.3 minutes of coding without interruption. LaToza *et al.* [6] found that 62% of developers found interruptions to be a serious problem. Parnin and Rugaber [8] report that for 80% of interruptions engineers are able to return to a programming task in 15 or fewer minutes. Interruptions when working on complex tasks are more difficult to return to compared to tasks that take less cognitive load [3]. Bailey *et al.* [1] found that interruptions negatively disrupted task performance, cognitive load, and put users in a negative emotional state. Self reporting can suffer from inaccurate memories of work, and in our work we measure actual time that developers spend on task.

At Google, Jaspan *et al.* [4] described their experiences in building out a system and quantitative logs pipeline called InSession that integrates dozens of development tools to understand developer behavior and productivity. They describe their design and implementation, how they validated the data, and show an application to understand the effect on readability. They also share key lessons learned. In contrast, we describe our work on quantifying and understanding developers workflows, their focus, and interruption patterns.

Work to clasify developer actions are extensive. For example, Ko et al. [5] analyzed developers' day-to-day information needs (n=17) by transcribing their behaviors through manual observation from 90-minute sessions in a two-months field study. They identified 21 information types and partitioned the observed work into work categories (coding, committing, debugging, reproducing, understanding, design reasoning, and maintaining awareness) and causes of task switching (face-to-face dialog, phone calls, instant messages, email alerts, meetings, task avoidance, getting blocked, task completion). In contrast, our work is based on analyzing large scale telemetry data from thousands of developers over a longer time horizon, logs at second level granularity, and quantifying time spent for various software engineering tasks.

Vasilescu *et al.* [9] gathered ecosystem-level data across GitHub about developers working on large collections of projects and developed models and methods to measure the rate and breadth of a developers' context-switching behavior. They found that the most common reason for multitasking is the interrelationship and dependencies between projects and that the rate of switching and breadth of a developer's work affect productivity (fewer projects per day led to higher productivity). In our work, we focus on better understanding the relationship between focus and interruptions, how tenure and centrality relates to focus, and the impact of an `auto-focus` system to automatically block interruptions.

Züger *et al.* [10] developed FlowLight to combine a physical traffic-light LED with an automatic interruptibility measure based on computer activity data to reduce the cost of in-person interruptions at work. They performed a multi-national field study (n=449) to evaluate their approach and found a 46% reduction in interruptions, increased awareness of the disruptiveness of interruptions, and positive self-rated productivity. In contrast, our work focuses on engineering workflows, utilizes telemetry data that is many times larger, and relates focus and interruptions to engineering productivity.

8 CONCLUDING REMARKS

In this work, we make the following contributions.

- We describe our large scale logging infrastructure powered by Workgraph that allows us to study how our engineers work with the goal of making their jobs more enjoyable and productive.
- We provide descriptive statistics on how long engineers remain focused at Meta. We find that the sum of time spent in personal-focus has a median 19.29 hours/week. When we restrict this time to flow sessions that are 12 or more minutes, we see a median of 14.25 hours/week. We also see that engineers have a median of 3 sessions that last longer than 1 hour for a median of 4.80 hours/week.
- We examine the factors that influence personal-focus time. We find that the median daily personal-focus for engineers

is increased when they perform more actions in IDE editors, with a feature importance of 0.48. The centrality of an engineer reduces their focus time (importance 0.37) as they coordinate more work. The remaining factors such as tenure, role, and pillar have little explanatory power with each having a feature importance less than 0.06, implying that personal-focus is consistent regardless of pillar/domain and engineering role.

- We study the impact of blocking Workchat notifications when a developer is in a deep coding focus. In a large experiment of `auto-focus`, we found that after engineers started using `auto-focus` to block interruptions after 12 minutes of coding, their median personal-focus time increased by 20.27%. We do not see a decrease in the number of messages sent. `auto-focus` has been rolled out to all employees at Meta.

`auto-focus` is only one example of work at Meta designed to reduce interruptions and increase focus. Future work at Meta includes investing in defragmenting calendars to make more focus time, by, for example, allowing engineers to marking meetings as “flexible" within a specific time frame. There are efforts to book time for code review, as well as to book time for coding on specific tasks. We are excited by the potential of making more time for engineering work, and hope our approach will inspire other projects and companies.

ACKNOWLEDGEMENTS

We would like to thank Elise Paradis and the Core Focus team for the work and definition of a focus session.

REFERENCES

[1] Brian P Bailey, Joseph A Konstan, and John V Carlis. 2001. The Effects of Interruptions on Task Performance, Annoyance, and Anxiety in the User Interface.. In *Interact*, Vol. 1. 593–601.

[2] Peter Cottle. 2021. On Programming Flow in a Remote-Work World. https://about.instagram.com/blog/engineering/on-programming-flow-in-a-remote-work-world.

[3] Mary Czerwinski, Eric Horvitz, and Susan Wilhite. 2004. A Diary Study of Task Switching and Interruptions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vienna, Austria) *(CHI '04)*. Association for Computing Machinery, New York, NY, USA, 175–182. https://doi.org/10.1145/985692.985715

[4] Ciera Jaspan, Matthew Jorde, Carolyn Denomme Egelman, Collin Green, Ben Holtz, Edward K. Smith, Maggie Morrow Hodges, Andrea Marie Knight Dolan, Elizabeth Kammer, Jillian Dicker, Caitlin Harrison Sadowski, James Lin, Lan Cheng, Mark Canning, and Emerson Murphy-Hill. 2020. Enabling the Study of Software Development Behavior with Cross-Tool Logs. *IEEE Software* Special Issue on Behavioral Science of Software Engineering (2020).

[5] Amy J. Ko, Robert DeLine, and Gina Venolia. 2007. Information Needs in Collocated Software Development Teams. In *29th International Conference on Software Engineering (ICSE'07)*. 344–353. https://doi.org/10.1109/ICSE.2007.45

[6] Thomas D. LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining Mental Models: A Study of Developer Work Habits *(ICSE '06)*. Association for Computing Machinery, New York, NY, USA, 492–501. https://doi.org/10.1145/1134285.1134355

[7] Andre N. Meyer, Earl T. Barr, Christian Bird, and Thomas Zimmermann. 2021. Today Was a Good Day: The Daily Life of Software Developers. *IEEE Transactions on Software Engineering* 47, 5 (2021), 863–880. https://doi.org/10.1109/TSE.2019.2904957

[8] Chris Parnin and Spencer Rugaber. 2011. Resumption strategies for interrupted programming tasks. *Software Quality Journal* 19, 1 (2011), 5–34.

[9] Bogdan Vasilescu, Kelly Blincoe, Qi Xuan, Casey Casalnuovo, Daniela Damian, Premkumar Devanbu, and Vladimir Filkov. 2016. The Sky is Not the Limit: Multitasking across GitHub Projects. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, Texas) *(ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 994–1005. https://doi.org/10.1145/2884781.2884875

[10] Manuela Züger, Christopher Corley, André N. Meyer, Boyang Li, Thomas Fritz, David Shepherd, Vinay Augustine, Patrick Francis, Nicholas Kraft, and Will Snipes. 2017. Reducing Interruptions at Work: A Large-Scale Field Study of FlowLight. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 61–72. https://doi.org/10.1145/3025453.3025662