The Work Life of Developers: Activities, Switches and Perceived Productivity

André N. Meyer, Laura E. Barton, Gail C. Murphy, *Member, IEEE*, Thomas Zimmermann, *Member, IEEE*, and Thomas Fritz, *Member, IEEE*

Abstract—Many software development organizations strive to enhance the productivity of their developers. All too often, efforts aimed at improving developer productivity are undertaken without knowledge about how developers spend their time at work and how it influences their own perception of productivity. To fill in this gap, we deployed a monitoring application at 20 computers of professional software developers from four companies for an average of 11 full work day *in situ*. Corroborating earlier findings, we found that developers spend their time on a wide variety of activities and switch regularly between them, resulting in highly fragmented work. Our findings extend beyond existing research in that we correlate developers' work habits with perceived productivity and also show productivity is a personal matter. Although productivity is personal, developers can be roughly grouped into morning, low-at-lunch and afternoon people. A stepwise linear regression per participant revealed that more user input is most often associated with a positive, and emails, planned meetings and work unrelated websites with a negative perception of productivity. We discuss opportunities of our findings, the potential to predict high and low productivity and suggest design approaches to create better tool support for planning developers' work day and improving their personal productivity.

Index Terms—Productivity, developer activity, work fragmentation, interruptions, human factors, user studies

1 Introduction

A software developer's work day might be influenced by a wide variety of factors such as the tasks being performed, meetings, interruptions from co-workers, the infrastructure or the office environment (e.g., [1], [2], [3]). Some of these factors result in activity and context switches that can cause fragmented work and that can have a negative impact on the developer's perceived productivity, progress on tasks, and quality of output (e.g., [4], [5]). As a result, researchers and practitioners have both had a long interest in better understanding how developers work and how their work could be quantified to optimize productivity and efficiency.

Researchers have investigated work practices and work fragmentation in detail from various perspectives, specifically the effect of interruptions on fragmentation (e.g., [6], [7], [8], [9]) and how developers organize their work in terms of tasks and working spheres (e.g., [5], [10]). Using both a diary and an observational study format to understand software developer work practices, Perry and colleagues gained several insights, including that most time was spent coding, and that there was a substantial amount of unplanned interaction with colleagues [2]. Singer and colleagues, using several study methods including tool usage statistics, found that developers spent most of their time reading documentation and that search tools were the most heavily used [3]. Since the time these earlier studies on developers' work practices were conducted, empirical studies of software development have focused more on particular aspects of a developer's work day. For example, Ko et al. observed software developers to determine what information was needed to perform their work and how they found that information [11]. Other studies have focused on how developers spend their time inside the Integrated Development Environment (IDE) (e.g., [12], [13]). The industry has also seen an increasing trend with self-monitoring tools to track activity and work habits, with applications such as RescueTime [14] or Codealike [15].

Starting in the 1970s, researchers have also been exploring various different ways to quantify a developer's productivity. Most of these identified productivity measures capture a small part or single aspect of a developer's work, such as the number of tasks per month [16], the number of lines of code written [17], or the resolution time for a modification request [18]. However, these studies on productivity are generally separate from studies of work fragmentation and of how developers work. Furthermore, these measures do not take into account the individual differences in development work that might affect productivity as pointed out by previous work [5], [19], [20], [21].

In this paper, we study developers' work practices and the relationship to the developers' perceptions of productivity more holistically, while also examining individual

- *A.N. Meyer and T. Fritz are with the University of Zurich, Zürich 8006, Switzerland. E-mail: {ameyer, fritz}@ifi.uzh.ch.*
- *L.E. Barton and G.C. Murphy are with the University of British Columbia, Vancouver, BC V6T 1Z4, Canada. E-mail: n4v9a@ugrad.cs.ubc.ca, murphy@cs.ubc.ca.*
- *T. Zimmermann is with Microsoft Research, Redmond, WA 98052. E-mail: tzimmer@microsoft.com.*

Manuscript received 9 May 2016; revised 21 Nov. 2016; accepted 11 Jan. 2017. Date of publication 22 Jan. 2017; date of current version 18 Dec. 2017. Recommended for acceptance by A.J. Ko
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2017.2656886

differences. In particular, our study seeks to answer the following research questions:

RQ1 What does a developer's work day look like?
RQ2 How fragmented is a developer's work?
RQ3 Are there observable trends in how developers perceive their productivity?
RQ4 What is the relationship between developers' activity and perceived productivity at work?

To investigate these questions, we designed and conducted a study involving the monitoring of 20 developers' interactions with their computer over a two week time period. From this monitoring, we were able to gather logs describing how a developer was interacting with the computer (i.e., through the keyboard or the mouse) and in what applications the interaction was occurring. Our monitoring also gathered self-reports from the developers about their current task(s) at 60 minutes time intervals, and a self-rating of their perceived productivity. The 20 developers from whom we gathered data worked for 4 different companies of varying size, with varying projects, project stages and customers, providing more diversity in our results than has been available in previous holistic studies. This approach also allows us to see whether earlier research findings, such as how much time developers actually spend coding [2] and typical coding related activities [22], hold in contemporary development, and to enhance emerging theories about fragmented knowledge work [10].

Based on our analysis of the gathered data, we observed that productivity is a highly personal matter and perceptions of what is considered to be productive are different across developers. No one model we built relating actions and activity to perceived productivity was able to explain a large number of developers. However, we did find that many developers consider email, planned meetings and work unrelated browsing as less productive activities, and usually perceive themselves as more productive when they have a higher user input rate as measured by mouse clicks and keystrokes. Further, developers' work is highly fragmented, as developers are spending only very short amounts of time (0.3 to 2 minutes) in one activity before switching to another one. Even though we observed that some aspects of a developer's work habits are highly individual we found consistent trends across multiple people. For example, some developers parcel their work out over a longer time span, while others choose to contain their work time and stay off of the computer during the evening. Some seem to be morning people, with higher productivity ratings in the morning hours, others are afternoon people. Finally, we discuss implications and opportunities of our findings to help improve and predict developer productivity.

This paper provides the following contributions:

- it provides insights into software developers' work habits, including the frequency and duration of performing particularly activities and application use;
- it provides data about the rhythms in developers' perceived productivity, which opens opportunities for retrospective tools and recommender systems for when developers might best perform particular activities;
- it demonstrates that productivity patterns for individuals are consistent, but vary when comparing across groups of software developers; and,
- it shows that perceived productivity and the factors that influence it, such as emails, meetings, or activity switches, are highly individual.

Section 2 presents relevant related work on developers' work practices, the high fragmentation of their work, approaches on quantifying development activities and on measuring productivity. Sections 3 and 4 describe the study method employed and the data collected. Section 5 presents the results of our study in terms of what a developer does, the fragmentation of a developers' work, the rhythms of a developer's perceived productivity and which activities and actions a developer perceives as productive. Section 6 outlines the threats to our results. Section 7 discusses implications and opportunities of the findings of our study for future tool support and presents results of a preliminary analysis on predicting two levels of productivity. Section 8 summarizes the paper.

2 Related Work

Related work can broadly be classified into four categories: developers' work practices, work fragmentation, the quantification of development activities and productivity.

2.1 Developers' Work Practices

Early studies that focus on a holistic perspective of how software developers spend their time at work, were conducted by Perry et al. and Singer et al. With two experiments, a diary-study and observations, Perry et al. found that by far most time was spent on coding, that there is much unplanned interaction with colleagues, and that work is generally performed in two-hour chunks [2]. Using a combination of surveys, interviews, observations, and collecting tool usage statistics, Singer and colleagues found that software developers spend most of their time searching for information as well as reading documentation and source code [3]. Most empirical studies since then have focused on various specific aspects of software development, such as the collaboration and communication of developers [23], the typical activities and tools related to coding [22], as well as developers information needs [11] and comprehension of software [24]. As an example, Goncalves and colleagues found in their observations and interviews that developers spend 45 percnet of their time collaborating, 32 percent for seeking information, and they only use software process tools during 9 percent of their time [23]. More recent studies by Minelli et al. [12] and Amann et al. [13] focused on how developers spend their time inside the IDE.

Our work confirms some of these findings, such as the little time developers spend in software development tools and on actual coding related activities. At the same time, it provides a more holistic and more complete picture of how a developer's work day looks across several companies and with detailed insights into how developers spend their work time on their computer (also outside the IDE), what programs they are using during their work, and how their work spreads over the whole day.

2.2 Work Fragmentation

One aspect of developers' work that has drawn much attention is the fragmentation of work. Developers work is fragmented and frequently interrupted, for instance, due to planned meetings, unexpected requests from a co-worker, unplanned or blocking tasks or even just background noise in the office. A large body of research has investigated interruptions—a major reason for fragmentation—in great detail, e.g., the length and types of interruptions, the frequency of self- versus externally initiated interruptions, their social context, the response time to certain types of interruptions, resumption strategies after an interruption, and their impact on work performance [6], [7], [8], [25], [26], [27], [28]. For example, by observing software developers and other information workers at work, Mark et al. found that 57 percent of all tasks are interrupted and are thus often fragmented into small work sessions [29] and Chong and Siino found that most interruptions—many of them were also self-initiated—lasted around 2 to 3 minutes [26]. While many of these interruptions are necessary, they can lead to a higher error rate, slow task resumption and an overall lower work performance [9], [30], [31], [32]. Parnin and Rugaber found, for instance, that only one out of ten interrupted programming tasks is being continued within a minute after the interruption [9].

Looking more broadly at the fragmentation of work, researchers have also investigated how developers organize their work in terms of tasks or working spheres (interrelated events that share a common goal), and the switching between these. In a study over a 7 month period, Gonzalez and Mark observed and interviewed software developers and found that there is a high level of discontinuity in developers' work, with an average of 3 minutes spent per task before switching, and an average of 11.5 minutes per working sphere [10]. In an extension of this study, Mark and colleagues also examined the influence of collocation and interruptions on the work fragmentation [29]. More recently, Sanchez and colleagues used interaction logs with an IDE to analyze fragmentation on software evolution tasks. Their analysis found that more interruptions and longer activity switches lead to a smaller edit ratio, indicating a lower productivity [33]. Further, they found that short breaks or interruptions between 3 to 12 minutes are most prevalent in developers' work with an IDE.

Supporting previous research on work fragmentation, we found that developers spend their time on a wide variety of activities and that they spend very little time in each one before switching to another. As one example, our results show that developers switch activities on average after less than two minutes except for meetings, which confirms the short switching times that González and Mark found [10]. Similarly, our results on developers having an average of 2.5 short breaks away from their computer per hour, with about 4 minutes of duration each, confirm the high frequency and impact of self- and externally initiated interruptions reported in other studies (e.g., [8], [26]). Our study extends previous research, by examining activities performed during a developer's work day from a more holistic perspective and by correlating them with perceived productivity.

2.3 Quantification of Work Activities

An increasing amount of people are using applications and wearable devices to track certain aspects of their lives, in particular related to physical activity and health [34], [35]. These applications provide users an opportunity to reflect upon their own activities as well as they support an improvement therein, such as a more active lifestyle [36], [37], [38], [39]. Along with this 'quantified self' movement of activity tracking, new applications such as RescueTime [14], TimeAware [40] and Hubstaff [41] have been developed to expand the self-quantification to work activities by automatically tracking and summarizing a worker's computer activity, sometimes even providing some productivity scores as in the case of RescueTime. A few of these applications, including Codealike [15] and Wakatime [42], also specifically target software developers and their work activities within the IDE.

In our study, we are also tracking all interactions of a developer with their computer, similar to but on a finer grained level than the mentioned tools and use the collected data to analyze developers' work, activities and practices and how it relates to their perceived productivity.

2.4 Quantification of Productivity at Work

Early on, researchers and practitioners explored multiple ways to quantify productivity of developers' work. Most of these productivity measures are based on a single artifact or deliverable over a time interval, for instance, the number of lines of source code (SLOC) written in a time interval [43], [44], [45], the number of function points per month [46], [47], the number of tasks completed per month [16], or the resolution time for modification requests [18], [48]. A more complete list of approaches to quantify productivity on the technical factors can be found in our previous work [5]. Most of these measures only capture a small part of a developer's work, also making it difficult to provide a more holistic picture of a developer's work and productivity [49]. The Personal Software Process (PSP) has taken this a step further by focusing on a set of basic metrics, such as time, size, quality, and schedule data, with the aim of improving an individual developer's skills and quality of work [19], [20], [50].

Researchers have also looked more broadly into the factors that affect the productivity of software development. In their systematic literature review, Wagner and Ruhe listed related work in a chronological order, starting in the late 1970s [51]. They categorized factors that influence productivity into technical (related to the product, process or IDE) and soft factors (related to the company and team culture, the developer's capabilities and experience and the work environment). DeMarco and Lister found, for instance, that the characteristics of a workplace, such as noise, privacy, and interruptibility can have a significant influence on a developer's performance for a given task [1]. As further examples, Boehm looked at factors such as the hiring of people and the use of language and tools, and their impact on improving development output [4] and Vasilescu et al. examined the influence of project-switching on productivity [21]. More recent studies have also looked more at soft factors on the individual level, such as the correlation of affective states and self-assessed productivity for programming tasks [52], or the impact of mood on performance for debugging tasks [53].

In a previous study, we took a first cut at investigating the trends and activities in developers' work with respect to their perceptions of productivity based on observations and

TABLE 1
Study Participants (IC: Individual Contributor, Distribution Is on a Seven-Point Likert Scale: Left = "Not at All Productive" (1), Right = "Very Much Productive" (7))

Participant			Total Dev.	Prof. Dev.	# Work	Perc. Prod. Ratings	
ID	Comp.	Role	Experien.	Experien.	Days	#	Distribution
S1	C	IC	15	10	8	45	
S2	C	IC	25	18	8	101	
S3	C	IC/Lead	23	16	9	62	
S4	C	IC	20	15	8	94	
S5	C	IC	19	15	8	40	
S6	C	IC/Lead	20	20	8	62	
S7	C	IC	16	11	15	80	
S8	C	IC	40	40	11	73	
S9	C	IC	29	29	12	89	
S10	C	IC	22.5	19	12	92	
S11	B	Lead	14	6	9	51	
S12	B	IC	1.5	0.5	7	40	
S13	D	Lead	23	12	10	76	
S14	D	IC	8	4	9	88	
S15	D	IC	5	1	9	71	
S16	A	IC	20	15	11	42	
S17	A	IC	19	5	17	62	
S18	A	Lead	17	17	16	53	
S19	A	IC/Lead	33	23	20	100	
S20	A	IC	8	7	13	30	
Average			18.9	14.2	11.0	67.6	

people's self-reporting [5]. This, however, left several questions unanswered, especially on what a developer's work day actually looks like, and how this relates to productivity over a longer period in time. We attempted to answer these lingering questions in this paper with a multiple-week field study of industrial software developers. Our work provides findings on various measures to model developer productivity and on the individual differences therein.

3 Study Method

To answer our research questions, we conducted an *in situ* study at four international software development companies of varying size. We collected data from 24 professional software developers using a combination of experience sampling (diary study) and a background monitoring application. The monitoring application logged a wide range of digital activities over several weeks with detailed precision. Experience sampling was used to collect participants' perceptions of productivity, as well as self-reported tasks and activities they performed throughout their work day.

3.1 Participants

We used personal contacts, emails and sometimes a short presentation at the company to recruit participants. Of the total 24 participants, we discarded the data of four participants as they did not respond to a sufficient amount of experience samples. Two participants responded to less than 5 experience samples over the course of the study, as they thought it was too intrusive for their work. The other two participants responded to very few samples since they were either working on a different machine as the one initially indicated or did not use their machine for more than an hour per work day.

Of the remaining 20 participants, 1 was female and 19 were male. All participants are professional software developers, with varying roles between individual contributors[1] and lead. At the time of the study, our participants had an average of 14.2 years (±9.6, ranging from 0.5 to 40 years) of professional software development experience and an average of 18.9 years (± 9.2, ranging from 1.5 to 40 years) of total software development experience, including education. An overview of our participants can be found in Table 1.

To capture various kinds of software development practices, we sampled developers from 4 different companies of varying size, in different locations and project stages, using different kinds of programming languages, and with different kinds of products and customers. Companies resided either in the USA (company A), Canada (company B and C) or Switzerland (company D). The company sizes varied from less than 10 developers (company D), to a few hundred (company C), and thousands of developers (company A and B). The project stages varied from working on initial releases (company D), over working on a next big release (company D and B) to being in a project maintenance cycle (company A and C). The developers in company A were mainly programming in C++ and C#, in company B in Java and C#, in company C in JavaScript, C# and SQL, and in company D in JavaScript, Java, C# and SQL. The products developed by the companies range from developer support tools, to power monitoring and robotics software, all the way to cloud-solutions for B2B and B2C customers.

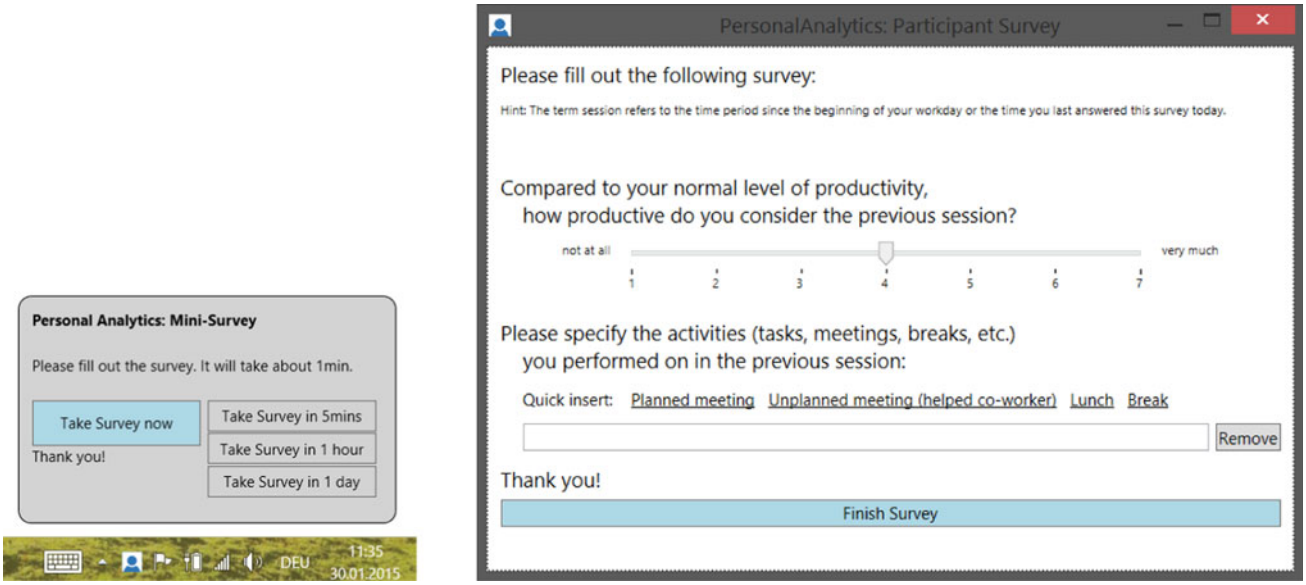
3.2 Procedure and Monitoring Application

The monitoring application was developed and tested to run on the Windows 7, 8 and 10 operating system. To make sure it works properly, the collected data is accurate and to optimize the performance, we deployed the tool in multiple steps. After an initial phase of extended testing on three researchers' machines, we deployed it to three developers in company B and one developer in company C over several weeks, to ensure correct functionality in many different computer set-ups and use cases and to ensure the tool is stable and reliable enough for daily use.

We then installed the monitoring application on the first day of the study after a presentation that included an introduction to the study and details on the data that was being collected with the monitoring application. Participants were assured of the anonymity and privacy of their data and were shown the location where the logged data was stored on their computer to give them full control over their data. Participants had the opportunity to censor parts of the collected data, which was reportedly done a few times, e.g., when participants were using their private e-banking. The monitoring application logged the currently active process and window title every 10 seconds, or an 'idle' entry in case there was no user input for longer than 10 seconds. In addition an event for each mouse click, movement, scrolling, and keystroke was logged. For keystrokes, we avoided implementing a key logger and only logged the time-stamp of any pressed key.

Perceived Productivity Self-Reports. To capture how developers perceive their own productivity, we used experience

1. We defined an individual contributor as an individual who does not manage other employees.



(a) Notification (b) Survey Pop-Up

Fig. 1. Notification to prompt participants to respond to the experience sampling survey.

sampling in the form of regular self-reports. Experience sampling has previously been used in multiple studies (e.g., [8], [54], [55], [56]), and allowed us to capture perceived productivity on a periodic and fine granular basis. For the experience sampling, a notification window appeared on the bottom right corner of the participants' main screen in regular intervals (see Fig. 1a) prompting the participant to answer a short survey on productivity. To minimize the risk of negatively influencing the participant's work, the notification window also offered options to postpone the survey. By default, the notification window was shown every 60 minutes, but participants could also change the interval, which was only done by one participant who changed it to a 90 minute time interval. Additionally, participants were also able to manually trigger the survey in case they wanted to answer the questions more often. If the participant was regularly working on a virtual machine or secondary device, we installed the application on the other devices as well, but disabled the pop-ups so as to not interrupt the participant more than necessary. Overall, participants answered 74.6 percent of the triggered self-reports within five minutes of the notification, 21.6 percent were postponed for an average of 38 ($\pm$43) minutes, and 3.8 percent were ignored.

Once the participant started the self-reporting, another window with the survey appeared (see Fig. 1b). This survey asked participants about their perceived level of productivity for the previous work session using a seven-point Likert scale and to specify the activities and tasks they performed. To facilitate the participant's response, the text boxes offered auto-completion and quick-insert buttons for frequently used and previously inserted activity descriptions. We only used a single question on productivity to minimize the disruption and also since previous research has shown that participants interpret different terms of productivity, such as efficiency, effectiveness, or accomplishment, very similar [56]. Since the survey questions remained the same throughout the study and were related to the current context, we expected the cognitive burden on the participant and the distraction for answering the survey to be relatively low, as also illustrated in other research [31], [57]. Participants used an average of 33.5 ($\pm$ 39.4) seconds to answer the two questions. We found no significant differences in the total number of times participants answered the survey per day over the whole course of the study, suggesting that they used similar effort throughout the study and the burden of answering did not increase for them.

Procedure. After we explained the study and installed the application, we asked participants to resume their normal working habits during the period of the study and answer the experience sampling probes when the pop-up appeared. We also told participants to ask us any questions about the study, the application or captured data at any point in time during the study. At the end of the study, we interviewed each participant to collect demographic information and information on the project they were working on, the company and their experience of using the monitoring tool and participating in the study. We then collected the monitoring and self-report data and un-installed the application. Table 2 summarizes the data we collected from the participants.

4 Data Collection and Analysis

During the study, the monitoring application collected data from 2,197 hours of participants' computer use over a total of 220 work days. Table 1 shows that each participant was part of the study for between 7 and 20 work days (mean: 11.0 days, $\pm$3.6). Table 2 shows how much of each type of information we collected. This section describes how we prepared the collected data for the analysis.

4.1 User Input Data

Whenever a study participant pressed a key, or clicked, scrolled or moved their mouse, the event-type and its timestamp were recorded by the monitoring application. We divided the events into work days, where a day spanned

TABLE 2
Data Collected by the Monitoring Application

Data	Description	Data Collected
Background Monitoring		
Program usage	current process name and currently active window title, captured once every 10 seconds	1,479,383 items
User Input		
Mouse clicks	as event happens	798,266 clicks
Mouse movement distance	aggregated pixels moved per second	2,248,367 entries
Mouse scroll distance	aggregated pixels scrolled per second	296,763 entries
Keystrokes	only count, not exact keystrokes (for privacy reasons)	3,515,793 keystrokes
Survey Pop-Ups		
Tasks and Activities	participants self-reported tasks and activities from the past 60 minutes (for one participant: 90 minutes)	2,237 items
Perceived Productivity	slider where the participant rates the perceived productivity of the previous work session	1,350 ratings

from 04:00:00 to 03:59:59 A.M. the following morning, very similar to Amann et al. [13]. This division was chosen as some participants stayed up late and logged input continuously before and slightly after midnight, suggesting that their 'work day' was not yet over at that time. With respect to time usage, but not application usage, we removed weekends, as we wanted to examine trends and based on initial inspection the weekend data was highly irregular. If a day had less than 10 minutes of recorded active input for a given developer, that day was not included when considering keyboard and mouse input.

Within a day, we determined *inactive* and *active* periods of computer use. An *inactive* period is a lapse in any type of input for 2 minutes or longer; conversely, an *active* period is 2 minutes or longer during which there is no inactivity. This 2 minute mark was chosen as a reasonable boundary based on previous research that found an average time of 2 min 11 sec being spent on any device or paper before switching [10], as well as research by Chong and Siino [26] who found that a typical interruption for one work team lasted 1 min 55 sec, and 2 min 45 sec for another team. We further defined inactive periods by sorting them into two categories: *short* and *long* breaks away from the computer. A short break from the computer is defined as a period of inactivity that lasts at least 2 minutes but less than 15 minutes and that is often spent with answering a co-workers question or as a coffee break. A long break from the computer is defined as a period of inactivity equal to or longer than the 15 minute threshold, and is often used for meetings and longer breaks from work, such as a lunch [33], [58]. We used a 15 minutes threshold based on previous work by Sanchez et al. that described a typical developer's coffee break to be 12 minutes on average and factored in a 3 minute deviation [33].

4.2 Preparing Program Data and Mapping to Activities

Our monitoring application recorded the current process and window titles of participants' program usage during their work day, once every 10 seconds. To provide a higher-level, aggregated view on participants' work, we mapped each entry of a program's use to activity categories, also taking into account the window titles for clarification. These activity categories group actions undertaken by a developer, for example the category *Coding* denotes any developer action that is related to reading, editing or navigating code. We reused the activity categories we identified with an open-coding approach in our previous study [5]. This mapping process was a semi-automated open-coding process; automated, where an action fit into an obvious category (e.g., SourceTree belonging to the activity category *Version Control*) and manual, where actions could not automatically be classified distinctively using the program and window title names (e.g., website names to the activity categories *Work Related Browsing* or *Work Unrelated Browsing*). When a participant did not switch programs or have any mouse or keyboard interaction for the past 10 seconds, the application logged it as 'idle'.

In order to complete the coding, we first defined a set of keywords for each activity that distinctly map a program to an activity. For instance, we mapped Microsoft Word to *Reading or Writing Documents* and Microsoft Outlook either to *Email* or *Planning*, depending on the details in the window title. We created a script using these keywords to produce the initial mapping, then inspected the results and iteratively refined the keywords until most programs were mapped. In cases where a program could be mapped to two or more activities, we performed a manual mapping. For example, as work in a text editor could be mapped to several activities (e.g., *Coding*, *Reading or Writing Documents*, or *Planning*), we manually mapped all editor use. Similarly, we mapped most website use manually, either to *Work Related Browsing* or *Work Unrelated Browsing*. In both cases, the window title held valuable information about the activity a developer was performing. We further manually checked all automated mappings for the following activities: *Debugger Use*, *Code Reviews*, and *Version Control*. All entries, related to coding, which could not distinctively be mapped to one of these categories, were mapped to *Coding*.

To better understand what participants were doing when they were not actively working on their computer ('idle' events), we combined the data logged by the monitoring tool with participants' self-reports. As most participants not only reported the tasks they worked on in the past work session, but also planned and informal meetings, and lunch and coffee breaks, we could in many cases infer if a period of 'idle' time belongs to one of these categories. In cases where the

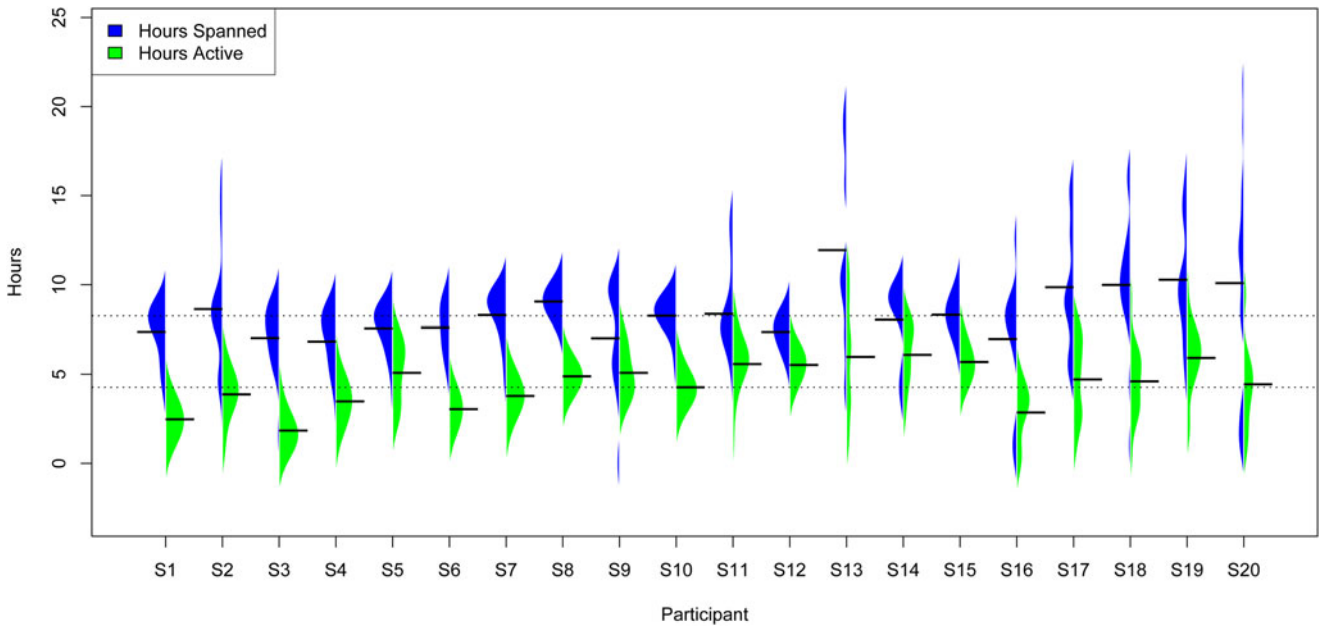


Fig. 2. Total hours of work versus hours active.

participant did not report a meeting or break, we had no way of identifying the reason for the 'idle' time, which is why the amount of time spent with planned and informal meetings might be higher than reported in this paper. The self-reports were not only used to map 'idle' time, i.e., time not actively spent on the computer, to breaks and meetings, but also to analyze developers' self-reported tasks.

The mapping algorithm is described in detail in the Appendix.

5 Results

This section presents the results of our study on developers' work practices and their relation to the perceived productivity by investigating the following research questions:

- *RQ1* What does a developer's work day look like?
- *RQ2* How fragmented is a developer's work?
- *RQ3* Are there observable trends in how developers perceive their productivity?
- *RQ4* What is the relationship between developers' activity and perceived productivity at work?

5.1 What Does a Developer Do?

To answer our first research question, "What does a developer's work day look like?", we analyzed the span of a developer's work, the amount of time during the span the developer was active, the nature of the breaks taken, the applications used, and the activities pursued.

5.1.1 Hours Spanned and Hours Active

The number of *hours spanned* by a developer's work day is defined as the time between the earliest and latest inputs on a given day, regardless of intervening activities or inactive periods. For example, if the first mouse click happened at 9:00 A.M. and the last keystroke happened at 17:30 P.M., the time span would be 8.5 hours. The number of *hours active* is the cumulative time when a participant was logging active input.

Fig. 2 contrasts the hours spanned and hours active per developer across all days monitored. Some participants (e.g., S13 and S20) tend to space their work out over as many as 21.4 hours, whereas others (e.g., S12 and S15) keep more compact work hours and remain active during the bulk of their time. Overall, developers averaged spans of 8.4 ($\pm$1.2) hours per day, with active time of 4.3 ($\pm$0.5) hours. It should be noted that the hours active are not synonymous with an individual's total working time; since the hours active value is based on the time the participant is using their mouse or keyboard, it does not account for meetings or other work activity away from the computer.

5.1.2 Short and Long Inactive Periods

Every hour, developers take an average of 2.5 ($\pm$0.8) short breaks that are about 4.2 ($\pm$0.6) minutes long each and in which the developers are not interacting with their computer. This results in a total of 10.5 minutes of inactive time every hour of work, which we assumed to be likely unplanned interruptions, such as co-workers asking a question or a quick coffee break. According to Minelli et al. [12], who analyzed how developers spend their time inside the IDE, inactive times are often spent with program comprehension. This notion of taking a few minutes to understand, think about or read the current artifact, such as code, a website or document, is likely another reason for these short inactivities. There was no obvious trend towards taking more or fewer short breaks during a particular part of the day; rather, short breaks appear to be fairly evenly distributed for all participants.

The number of long breaks in which developers did not interact with their computer for longer than 15 minutes averaged 3.3 ($\pm$1.4) per day, with a total length of 54.7 ($\pm$28.2) minutes; this corresponds to an expectation of two longer coffee breaks, a lunch, and perhaps a meeting. The participants with a high number of long breaks appear to be those who have a tendency towards longer hours spanned, with additional long breaks happening in the late afternoon or evening before these individuals returned to work. Since

TABLE 3
Top 10 Used Applications (Sorted by Usage)

Application	% of time used	# of users
Microsoft Outlook	14.2%	18
PuTTY	12.8%	8
Google Chrome	11.4%	16
Microsoft Internet Explorer	9.4%	20
Microsoft Visual Studio	8.3%	13
File Explorer	6.6%	20
Mozilla Firefox	5.9%	8
Eclipse	3.0%	10
Microsoft OneNote	2.3%	9
Command Line	2.2%	16

these long breaks were likely planned by the developers, they were also more likely to be self-reported as a distinct activity in the pop-up surveys.

5.1.3 Applications Used and Frequency of Use

Participants used a total of 331 different applications, with each participant using an average of 42.8 (±13.9) different applications over the study duration and 15.8 (±4.1) applications per day.

Table 3 shows the ten most popular applications across all participants (all were using Windows operating systems). There is a notable amount of time spent using File Explorer, although this may be due to the fact that it is the only application, other than Internet Explorer, that was used by all 20 participants. Despite everyone using Microsoft Internet Explorer, 80 percent of the participants also used Google Chrome and spent more time in it than in the Internet Explorer.

5.1.4 Activities Pursued

Table 4 shows the activities developers pursue during their work days. A developer's typical work day is mostly spent on coding (21.0 percent), emails (14.5 percent), and work-related web browsing (11.4 percent). Using the debugger, reviewing other developers' code, and version control account for just 2.4 percent of the participants' time. When looking at individual versus collaborative activities, 24.4 percent of a developer's day is spent pursuing collaborative activities with co-workers, customers, or managers, such as planned or ad-hoc meetings and emails. These percentages do not include uncategorized inactive time towards the total time as the monitoring application could only capture non-computer work in case the participants self-reported it.

When we inspected the data, we observed a notable range between the minimum and maximum time developers spent on each activity per day. The minimum time was virtually 0, or only a few seconds per day. The maximum time a participant spent on a single activity was 12.8 hours on one day: S13, who was evaluating various continuous integration and build systems. It is clear that the duration and type of activities vary greatly depending on the individuals and their current goals. We also observed differences in the distribution of the activities between companies. For example, developers S11 and S12, both from the same company, spent significantly less time on emails, on average just 0.7 minutes (±0.5) per day, compared to the other participants, who spent an average 74.3 minutes (±74.8) on emails.

The amount of time spent in coding or debugging might be higher than reported, as it was not possible to map all activities from the category Remote Desktop (*OtherRdp*), as there was not enough context available for an accurate mapping either automatically or manually. Similarly, the

TABLE 4
Developers' Fragmented Work: Activities Performed

Activity Category		% of time over whole period	Duration per day (in hrs)			Time spent before switching (in mins)		
			Avg	Stdev	Max	Avg	Stdev	Max
Development								
Coding	reading/editing/navigating code (and other code related activities)	21.0%	1.5	±1.6	7.3	0.6	±2.6	135.7
Debugger Use	using the debugger inside the IDE	0.4%	0.1	±0.2	0.8	0.5	±0.8	13.4
Code Reviews	performing code reviews	1.3%	0.3	±0.4	2.1	1.3	±4.5	13.4
Version Control	reading/accepting/submitting changes	0.7%	0.1	±0.3	2.2	0.6	±1.0	12.9
Email	reading/writing emails	14.5%	1.1	±1.3	8.1	0.9	±4.8	89.6
Planning	editing work items/tasks/todos; creating/changing calendar entries	4.8%	0.5	±1.1	5.1	1.1	±2.5	67.5
Read/write documents	reading/editing documents and other artifacts, e.g., pictures	6.6%	0.5	±0.7	4.5	0.8	±3.3	114.7
Planned meeting	scheduled meeting/call	6.5%	1.0	±1.3	7.1	15.8	±35.3	203.1
Informal meeting	ad-hoc, informal communication; e.g., unscheduled phone call / IM, or colleague asks a question	3.4%	0.5	±0.6	4.2	2.0	±6.5	138.2
Work related browsing	Internet browsing related to code/work/task	11.4%	0.8	±1.3	12.8	0.5	±5.5	102.6
Work unrelated browsing	Internet browsing work unrelated	5.9%	0.5	±0.7	3.4	1.1	±4.3	91.8
Other	Anything else; aggregates several small sub-categories, such as changing music, updating software, using the file explorer or having a break	11.4%	0.8	±1.4	10.5	0.4	±5.6	112.5
Other RDP	Remotedesktop use which could not be mapped to another category	12.0%	1.5	±1.8	8.2	0.3	±2.6	85.4

amount of time spent in planned and informal meetings might be higher than reported, as participants likely did not self-report all meetings they attended via the pop-up survey. This made it impossible to map all 'idle' time to an activity.

5.1.5 Developers' Self-Reported Tasks

An analysis of developers' self-reported tasks shows that there is a wide variety in what developers work on, but in particular also, in what developers denote as a task. In many cases, participants reported activities they were performing, such as "coding", working on "emails" or performing a "web search", rather than the intention of the activity, such as the change task or the bug they were trying to fix. Only in very few cases, did participants mention a bug ID on which they were working. Furthermore, reported and worked on tasks varied in their granularity. While some participants were working on a task only a single time for a part of the 60 to 90 minutes time window, others reported to work on the same task for several days.

Overall, due to this variance in task definition and granularity, the self-reported tasks did not provide much further insights into developers' work days other than help with disambiguation of our mappings in some cases. Also, while the number of resolved or worked on tasks has been rated as a relatively good measure for assessing one's own productivity [5], the variance in self-reported tasks, especially also across developers, suggests that it might be a somewhat individual help for assessment at best.

5.2 How Fragmented Is the Work?

To answer our second research question, "How fragmented is a developer's work?", we analyze how much time they spend on an activity before switching to the next one. The last three columns of Table 4 present the times a developer pursues each one of the activities before switching to another one. With the exception of planned meetings, a developer only remains in an activity between 0.3 ($\pm$2.6) and 2.0 ($\pm$6.5) minutes before switching to another one. These very short times per activity and the variety of activities a developer pursues each day illustrate the high fragmentation of a developer's work. The low standard deviations for each activity further emphasize this finding, which is similar to Mark et al.' s, Ko et al.'s and our previous observations [5], [10], [11]. At the same time, our data also suggests that there are exceptions to this high work fragmentation and that in rare occasions, developers spend long hours without switching activities. For example, participant S4 was coding in the late afternoon for 135.7 minutes, without any break longer than 2 minutes. Planned Meetings are the only exception to the short time periods spent on a single activity with an average duration of 15.8 minutes ($\pm$35.3) before switching. Our analysis of the data also suggests that developers are not using their computer in most of these planned meetings. The opposite is true for informal meetings, for which our monitoring tool recorded user input every few minutes. It is important to note that an activity switch, while contributing to work fragmentation, is not necessarily a task switch. A developer might switch activities several times while working on the same task, e.g., switching from coding in the IDE to the web browser to search for API documentation or code snippets.

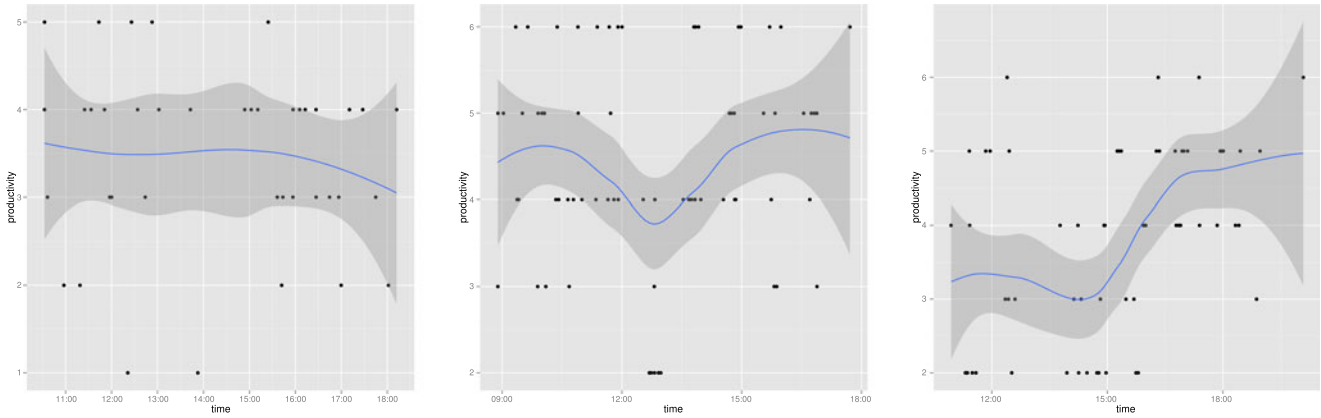
To better understand the high number of activity switches, we performed an n-gram analysis to identify the activities which developers often perform in a sequence together. The activity pattern, which occurred most often, was a quick switch to emails during coding tasks. This finding is supported by our results in a previous observational study, where we learnt that developers often perform very quick and short context switches during waiting times, which increases their perceived productivity [5]. Similarly, Amann et al. found that developers continue working while builds run in the background [13]. Developers also regularly switch away from coding to work related web browsing (22.1 percent), reading or writing documents (14.3 percent) or planning (14.2 percent). These switches can be explained with the search for additional information necessary to complete a task, such as a task description from an email, a quick research on the web (e.g., for a code snippet or tutorial), or reading a documentation-file. After these quick switches, developers usually switch back to their main coding task.

We were also interested in better understanding what activities developers were performing before they were interrupted by a co-worker, to learn where to focus for building task resumption tools. When developers switched their activity to an informal meeting, they were emailing in 40.1 percent of the cases, coding in 18.1 percent of the cases, and browsing the web (work related) in 13.5 percent of the cases before the switch. Switches to work unrelated web browsing were most often caused during coding tasks (35.5 percent) and during work related web searches (26.7 percent), likely to get a quick break from work.

5.3 Perceived Productivity Changes?

Our third research question asks "Are there observable trends in how developers perceive productivity?". We use the developers' self-ratings of their productivity via the pop-up survey to investigate this question. For each participant, we plot the perceived productivity ratings against the time of day the rating was collected; thus, all ratings from an individual are condensed across the hours of the day in one plot.

From an analysis of these plots, we found that although there was a lot of variation between individuals, the plots can be categorized into three broad groups: morning people, afternoon people, and those whose perceived productivity dipped around lunch. Fig. 3 shows examples of these three types. The curved regression line in the figures shows the overall pattern of what part of the day an individual developer typically felt more or less productive with the shaded area showing the confidence range. Morning people were rare in our sample set (20 percent of all participants); Fig. 3a shows S5's perceived productivity pattern, which is our clearest example of the trend but is not very pronounced. Afternoon people (8 out of 20, 40 percent) may be those who are industrious later in the day, or that feel more productive as a result of having the majority of their work day behind them (Fig. 3c). The greater number of afternoon people in our sample reflected previous research that showed that information workers perceive themselves as most productive in mid-afternoon, peaking around 2-3 P.M. [59]. The low-at-lunch group (35 percent) may see long breaks as unproductive, or they may simply lag in



(a) Morning Person (b) Low at Lunch Person (c) Afternoon Person

Fig. 3. Three types of developers and their perceptions of productivity over the course of a work day.

effectiveness as their physical processes draw focus away from work (Fig. 3b).

These graphs and numbers suggest that while information workers in general have diverse perceived productivity patterns, individuals do appear to follow their own habitual patterns each day. Only for one of the twenty participants it was not possible to determine a dominant category.

5.4 What Are Productive Activities?

To answer our fourth research question, "What is the relationship between developers' activity and perceived productivity at work?", we built explanatory models relating the action and activity data to the productivity ratings.

The purpose of the *explanatory models* is to describe which factors contribute to the productivity ratings reported by the study participants. For each participant, we built one stepwise linear regression model for a total of 20 models. We chose linear regression because it is a simple and intuitive way to model data. The dependent variable is the reported productivity rating and the independent explaining variables are: (1) session duration, (2) number of certain events such as activity switches, (3) keystrokes per minute, (4) mouse clicks per minute, (5) amount of time spent in activities normalized by session length, and (6) how much of a session was before mid-day (noon) in percentage.[2] By choosing linear regression, we assume that the productivity ratings are interval data meaning that the distance between the productivity ratings 1 and 2 is the same as the distance between the ratings 2 and 3, and so on. To facilitate comparison across models, we specified the intercept value for all models as 4, which corresponds to an average perceived productivity.

Table 5 shows the results of the explanatory modeling. Each column corresponds to the perceived productivity model of a participant and each row corresponds to a factor in the model. To reduce the complexity of the table, we only report the sign of the coefficients; the full coefficients are available as supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2017.2656886 [60]. A plus sign (+) in a cell indicates that a factor has positive influence in a model; for instance, S1 reported higher productivity ratings with a higher number of self-reported tasks. Similarly, a minus sign (−) indicates negative influence; for instance, S3 reported lower productivity ratings for higher session durations. Empty cells correspond to variables that either were removed as part of the stepwise regression, or were not statistically significant. An NA value indicates that an event or activity did not occur for a participant in the study period; for instance, the NA for S1 in *Debugger Use* means that S1 never used the debugger in the IDE during the study period. In a few cases, we were also not able to map all 'idle' log entries to the two activity categories of informal meetings or planned meetings due to a lack of information provided in the self-reports. These cases are also denoted with NA.

Based on the results presented in the table we can make several observations:

(i) No two explanatory models are the same. This suggests that productivity is a highly personal matter and that perceptions of what is considered to be productive are different across participants.

(ii) No single factor provides explanatory power across all participants. Furthermore, the same factor can have positive influence for one participant, and negative influence for another participant, for example the Number of Self Reported Tasks has both negative (2×) and positive influence (2×).

(iii) The Number of Keystrokes and the Number of Mouse Clicks have more often positive influence (7×) than negative influence (1× and 2× respectively).

(iv) The activities Email (5× negative), Planned Meeting (6× negative), Work Unrelated Browsing (5× negative), and Idle (6× negative) have more often negative influence.

5.5 Summary of Results

Our analysis provides a broad range of insights on the relationship between developer's work practices and activities and their perceived productivity. Table 6 summarizes some of the key findings.

6 Threats to Validity

The main threats to our study are construct validity threats due to the monitoring application we used to collect data. These, and the threats to internal and external validity, are described in this section.

2. We chose mid-day, since a previous study found differences in knowledge workers' activities before and after mid-day [59].

TABLE 5
Explanatory Productivity Models for Participants ('+' Indicates Positive, '-' Negative Influence; 'NA' Indicates a Never Performed Activity)

Neg	Pos	NA	Particpant	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20
			Ratings (total)	45	101	62	94	40	62	80	73	89	92	51	40	76	88	71	42	62	53	100	30
			Ratings (discarded)	0	29	3	4	0	0	0	7	10	10	1	0	7	0	0	0	4	8	11	4
			Ratings (included in model)	45	72	59	90	40	62	80	66	79	82	50	40	69	88	71	42	58	45	89	26
			Ratings (distribution)																				
3	3	0	**Session Duration (in hours)**		+	−	−		+		−					+							
2	1	0	**Percent of Session Before Noon**											−			−				+		
			Per Minute																				
2	2	0	# SelfReportedTasks	+								−		−		+							
2	4	0	# ActivitySwitches		+		−		+				+	−				+					
1	1	4	# Meetings	NA		NA				+				NA				NA	−				
2	1	7	# InstantMessagingSwitches	NA	NA	NA	NA	+		NA	NA		NA								+		−
1	**7**	0	# KeyStrokes					+						+		+	−	+		+	+		+
2	**7**	0	# MouseClicks	+		+				+		+	+	+	−		+				−		
			Percent Activity																				
4	1	0	DevCode						−				−			−					+		−
0	1	9	DevDebug	NA	NA	NA				NA	NA	NA	NA	+							NA		NA
2	1	12	DevReview	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	−			NA	NA				−	+
2	0	3	DevVc		NA		−		NA				−							NA			
5	0	0	Email							−			−	−		−						−	
2	3	0	Planning		+		+							−	+							−	
4	3	0	ReadWriteDocument	+		+			−				−	+		−				−			
6	0	5	PlannedMeeting	NA		NA		−	−	−		−		NA	NA	−		NA					−
3	2	9	InformalMeeting	NA	+	NA	NA	NA					−	NA	NA	−	NA	NA		NA	+		−
2	0	8	InstantMessaging	NA	NA	NA	NA	NA	−	NA	NA		NA								−		
1	2	0	WorkRelatedBrowsing							−			+								+		
5	0	2	WorkUnrelatedBrowsing					−			NA		−			−	−	−		NA			
2	1	0	Other								+						−						−
2	1	4	OtherRdp	NA		NA		−			+						NA	−					NA
6	2	2	Idle		−		−	−	−		+	−		NA		−	+	NA					

Columns 'Neg' | 'Pos' count the number of times a variable had negative | positive influence. The ratings are distributed on a 7-point Likert scale: left = "not at all productive" (1), right = "very much productive" (7).

6.1 Construct Validity

The main threat comes with the metrics we base our analysis on, as it is limited to the data we collected with our monitoring application. We believe that a chronological record of application use, user inputs, and the addition of self-reported productivity, tasks, and activities provide a reasonable basis to analyze a developer's work. Though, we cannot exclude the possibility that any other factors influence a developer's work day and productivity.

Running a monitoring application in a real-world scenario might capture inaccurate data, due to bugs in the logging application or stability issues. To mitigate this risk, we ran several test-runs in different scenarios prior to the study, and observed a user for several hours to compare the logged with the observed data. No major problems with the tracker were reported during the tests or at the time of the study.

Even though the monitoring tool is able to capture a broad range of activities on a participants' computer, it does not capture activities away from the computer. Therefore, we asked participants to record their activities/tasks for their time away from the computer in the periodic self-reports. Furthermore, to capture activities performed on secondary computers or remote desktops, we asked participants to install the monitoring application without the self-reporting feature on these machines as well.

Understanding, categorizing and analyzing the data poses another threat to validity, especially since it is not straightforward to identify all activities from the collected

TABLE 6
Summary of Some of the Study Key Findings

#	Finding	Section
F1	Developers only spend about half their time active on their computer.	5.1.1
F2	For every work hour, developers have an average of 2.5 short breaks, totaling 10.5 minutes of unplanned time away from their computer.	5.1.2
F3	Developers spend about a fourth of their time on coding related activities and another fourth of their time on collaborative activities.	5.1.4
F4	The range and time spent on activities varies greatly depending on the individual and company.	5.1.4
F5	Developers' work is highly fragmented, spending very short amounts of time (0.3 to 2 minutes) in one activity before switching to another one.	5.2
F6	Developers' perceived productivity follows habitual patterns, broadly categorisable as morning people, afternoon people and "low-at-lunch" people.	5.3
F7	Productivity and the factors that influence it are highly individual.	5.4
F8	The number of mouse clicks and key strokes often have a more positive, email, planned meetings, and work unrelated browsing a more negative impact on perceived productivity.	5.4

data. For instance, mapping 'idle' times to self-reported breaks, planned meetings and informal meetings could not be automated. We also needed to discard outliers, such as very short work in the middle of the night or on weekends. To mitigate this risk, we did a manual mapping of activities we were uncertain about and checked random samples of the semi-automated mapping. Assumptions made and thresholds defined were carefully discussed, based on previous related work, and described in the paper in detail. The short interviews at the end of the study further helped us to interpret each participants' data and work behavior.

6.2 Internal Validity

Running a monitoring application on participants' computers might pose privacy concerns to participants. To address these concerns, we tried to be very transparent about the data we collected. The participant was shown the location where the logs were saved and given the opportunity to censor them. We did not collect information about what was being typed or clicked on, merely that these events were occurring. We also assured the participant that all collected data will be anonymized and saved on password-protected devices or in locked filing cabinets.

Monitoring participants' during their work bears the risk of changing their behavior. To mitigate these risks, we tried to collect as much data as possible in the background and optimized the performance of the data collection to avoid lags, creating a non-intrusive experience for participants. Several participants explicitly mentioned that they usually forgot about the monitoring application and were only reminded when they were prompted about the self-reports.

Interrupting participants with a short survey once an hour might have influenced their work behavior and habits. To address these concerns, we tried to only show the pop-ups when necessary and reduce the effort needed to fill them out by showing previous responses, having quick response buttons, and auto-completion boxes. Additionally, the participant had the chance to postpone the survey in case it interrupted at an inopportune moment. The continuously very short but stable amount of time used to answer the periodic survey throughout the study and the small variation in the number of responses per participant and day also suggests that participants' behavior was not affected much and that they did not get annoyed by the survey.

6.3 External Validity

The number of participants or the selection of participants might limit the generalizability of the results of this study. In particular, our participants were all using Windows as their operating system due to our monitoring application being built for Windows. Overall though, we tried to mitigate the threats to external validity and generalizability by selecting participants from four different software companies of varying size, with more and less well-established products, different kinds of customers, and different stages of product development. Studies were spread to three different countries, Canada, US, and Switzerland, and across half a year. Additionally, all participants are professionals who were studied in their everyday, real-world work environment and not in an experimental exercise. The external validity is further strengthened by the broad range of development experience of our participants, ranging from junior to senior developers with an average professional development experience of 14.2 years ($\pm$9.6, ranging from 0.5 to 40 years). Finally, our participants worked on projects using 7 of the top 10 most used programming languages according to a recent large-scale study [61]. While the large-scale study also showed that 55 percent of all developers use Windows as an operating system and our focus on Windows thus maps to a majority of developers, further studies with a broader participant pool are needed to assess the generalizability of our results.

Another limitation might be the study running for roughly two weeks per participant, as developers' activities might vary greatly at different stages and iterations of their projects. We tried to mitigate this risk by having participants from different teams at different stages of their project and iterations, and by staggering the monitoring period between participants so that varying times of the year were covered. In the final interview, most participants also agreed that the study took part in fairly usual, and not extraordinary, work weeks.

7 Discussion

The results of our study shed new light on the work practices, fragmentation and perceptions of productivity of individual software developers. In the following, we discuss implications and opportunities of our findings, in particular the individuality of productivity and its use for designing better tool support, and we report on an exploratory analysis to predict high and low productivity sessions.

7.1 Individuality of Productivity

To quantify a developer's productivity, related work predominantly focused on a single or a small set of outcome measures, such as the lines of code or function points. While these measures can be used across developers, they neglect to capture the individual differences in the way that developers' work as well as the differences in their work and their perceived productivity. The results of our study show that perceived productivity is in fact a very personal matter and the influence and impact of the factors used for our explanatory productivity models varied greatly. This suggests that measures or models of productivity should take into account the individual differences in what is perceived as productive or not and capture a developer's work more holistically rather than just by a single outcome measure. Such individual models could then be used to provide better and more tailored support to developers, for instance, to foster focus and flow at work.

At the same time, our results also show that while there are individual differences, there are tendencies amongst groups of software developers, for instance, with the number of key strokes and mouse clicks having a positive influence on productivity perception for seven of the 20 participants. Similarly, we identified types of developers with similar trends of perceived productivity over the course of the day, including morning and afternoon people, which resemble the morningness and eveningness types that Taillard et al. identified in their large-scale study on people's sleeping habits [62]. These results suggest that it

might be possible to identify clusters of software developers with fairly similar productivity models despite individual differences, which could then be used to provide tool support tailored to these clusters, for instance, for scheduling a productive work day.

7.2 Supporting Flow and Retrospection

In our previous study, we found that developers feel particularly productive when they get into "the flow" without having many switches [5]. Results from this and other studies suggest that getting into the flow during work might not be very easy, given the high fragmentation of work and the many short breaks and interruptions.

At the same time, our analysis of the collected data suggests that it might be possible to identify individualized proxies for developer productivity, such as using the number of mouse clicks or key strokes per minute or the time spent in work-unrelated browsing for certain developers. Knowing if a developer is productive or unproductive at the moment by using such proxies could be used to support getting and staying in a highly productive "flow" state. In particular, one could use this to indicate the availability of a developer for interruptions by changing the availability status in instant messaging tools, or with a visual external cue to avoid external interruptions at particularly inopportune moments, similar to what Züger and Fritz suggested [63]. Or, one could also use this to provide awareness to the developers themselves on their flow and productivity, by indicating them when they are stuck and it might be time to ask a co-worker for help or to take a break, or even blocking work-unrelated websites for 25 minutes—similar to the Pomodoro technique [64]—and helping them to focus when they are procrastinating.

Being able to predict, to some extent, a developer's productivity on an individual basis could also be used to provide developers with individualized retrospection support, in the form of daily or weekly summaries of their work. With the *Quantified Self* movement, more and more people are using applications and devices to track themselves—mostly with a focus on the non-work related activities, such as sports. These persuasive technologies provide users an opportunity to reflect upon their own activities and support desired improvements, such as a more active lifestyle (e.g., [34], [35], [39]), due to self-monitoring and goal-setting [39]. A proxy of an individual developer's productivity might provide such benefits for the software development domain, by increasing developers' awareness about their own work practices and productivity and thereby helping them to improve them.

7.3 Scheduling a Productive Work Day

By knowing the trends of developers' perceived productivity and the activities they perceive as particular productive/unproductive, it might be possible to schedule the tasks and activities developers must perform in a way that best fits their work patterns. For example, if a developer is a morning person and considers coding particularly productive and meetings as impeding productivity, blocking calendar time in the morning for coding tasks and automatically assigning afternoon hours for meeting requests may allow the developer to best employ their capabilities over the whole day. Or, it could remind developers to reserve slots for unplanned work or interruptions at times where they usually happen.

7.4 Predicting High & Low Productivity

To examine whether we might be able to identify high and low productivity sessions with the collected data, we performed an initial, exploratory analysis, building *predictive models* using logistic regression. For each participant, we computed the median productivity rating individually, which we assumed to be the standard perceived productivity of a developer. We then used the productivity (high and low) as the dependent variable and the factors used in the explanatory models (see Table 5) as the independent variables, and we built two prediction models using binary logistic regression:

Model 1: Is the reported perceived productivity *above* the median productivity? (High Productivity)

Model 2: Is the reported perceived productivity *below* the median productivity? (Low Productivity)

We built and evaluated the models for each participant using 50 random split experiments: 2/3 of the participant's data was used for training and the remaining 1/3 of the data was used for testing the model. In total, we ran $50 \times 2 \times 20 + 50 = 2{,}050$ experiments. For each experiment, we measured the success of the predictions with precision and recall. Precision represents how many of the returned results are relevant (correct), and recall represents how many of the relevant results were returned. We then averaged the precision ratings over the 50 experiments for each model and participant to receive a single precision rating. We did the same for recall.

Table 7 shows the results. In addition to the precision and recall values, we report in the Ratio columns the percentage of High productivity and Low productivity sessions for each participant. On average, the models have a precision of 0.57 (High Productivity) and 0.56 (Low) and recall of 0.48 (High) and 0.43 (Low). For some participants, the precision and recall values are above 0.70. The results are promising and suggest that even with a relatively small number of reported productivity ratings, it is possible to build personalized, predictive productivity models. To build the models we used the session length and information about events (keystrokes, mouse clicks), and activities. We expect that with more context and data, such as active task information, window contents, calendar information, development experience, time of day, or possibly biometrics, the quality of the predictions can be improved further.

7.5 Privacy Concerns

As with all approaches that collect personalized data on people, collecting information on a developer's activities on the computer potentially raises many privacy concerns. Especially given the focus of our study on productivity, some people were skeptical and declined to participate. This indicates the sensitivity of the data and the need for further research on the privacy concerns of such broad, work-related data. We believe that integrating potential users early on in the design process of building such a tool is crucial to increase acceptance of and engagement with

TABLE 7
Models to Predict High/Low Productivity Sessions

	Ratings	High Productivity			Low Productivity		
Partic.	Distr.	Ratio	Prec.	Recall	Ratio	Prec.	Recall
S1		0.36	0.64	0.39	0.24	0.85	0.03
S2		0.54	0.76	0.71	0.15	0.41	0.46
S3		0.36	0.69	0.48	0.39	0.54	0.58
S4		0.13	0.46	0.11	0.39	0.62	0.33
S5		0.13	0.55	0.27	0.45	0.53	0.47
S6		0.27	0.32	0.43	0.26	0.56	0.59
S7		0.38	0.63	0.64	0.26	0.36	0.11
S8		0.03	0.60	0.42	0.02	0.52	0.68
S9		0.41	0.41	0.43	0.27	0.62	0.38
S10		0.07	0.17	0.41	0.09	0.56	0.48
S11		0.36	0.60	0.62	0.26	0.48	0.42
S12		0.50	0.71	0.76	0.50	0.67	0.62
S13		0.41	0.59	0.62	0.45	0.54	0.52
S14		0.49	0.60	0.64	0.27	0.59	0.45
S15		0.46	0.67	0.77	0.18	0.64	0.18
S16		0.29	0.54	0.40	0.40	0.73	0.52
S17		0.34	0.61	0.52	0.22	0.29	0.13
S18		0.18	0.47	0.40	0.47	0.54	0.49
S19		0.25	0.72	0.08	0.43	0.48	0.54
S20		0.50	0.67	0.52	0.50	0.60	0.62
Average			**0.57**	**0.48**		**0.56**	**0.43**

the tool. Furthermore, we expect that the voluntary use of such applications and its ability to tailor to the individual is important for its success since it focuses on the intrinsic motivation of developers to improve or better understand themselves. Requiring the use of such tools by upper management on the other hand will lead to a 'gaming' as previous research found and suggested (e.g., [49]), since developers might fear that the gathered information could influence their employment and increase pressure.

8 Summary

Related work has proposed a wide variety of approaches to quantify the productivity of software developers, mostly only taking a single aspect of a developers' work into account. In our paper, we present a more holistic approach to examine developers' work practices and their relation to perceived productivity. We conducted a study with 20 professional software developers in four different companies *in situ*, to investigate how developers spend their work days and what activities they perform. We applied a combination of computer logging and experience sampling by using a background monitoring application and by letting developers answer pop-up questions every 60 minutes. The results show that developers spend their time on a wide variety of activities, about a fourth of it on collaborative activities, such as meetings or emails. On average, participants spend only between 0.3 and 2 minutes on each activity, before switching to the next one. They also have 2.5 short per hour and 3.3 breaks per day. This demonstrates how fragmented a developer's normal work day is.

Based on developers' self-reports, we analyzed how their perceived productivity changes throughout a work day. We found a lot of variation between individuals, but that they can roughly be grouped into morning people, low-at-lunch people and afternoon people. We also correlated perceived productivity with activities and user input using a stepwise linear regression model per participant. The data suggested that productivity is a personal matter and that perceptions vary greatly as different factors in a developer's work day can influence productivity either positively or negatively. More user input was often associated with a positive, while emails, planned meetings and work unrelated websites were most often associated with a negative perception of productivity. Based on our findings, we propose a number of design approaches and tools to help increase developer productivity. For instance, by supporting developers to get into and stay in "the flow", by reducing interruptions at inopportune moments and by helping them to focus when they are procrastinating. Finally, we ran an exploratory analysis of predicting productivity for individuals, based on their computer usage. The results are promising and suggest that even with a relatively small number of reported productivity ratings, it is possible to build personalized, predictive productivity models. In the future, we plan to work on improving the quality of these predictions by including more context and data, such as active task information, experience, time of the day, and biometrics.

Acknowledgments

The authors would like to thank the study participants and all of our reviewers for their insightful remarks. This work was funded in part by ABB, SNF and NSERC.

References

[1] T. DeMarco and T. Lister, "Programmer performance and the effects of the workplace," in *Proc. 8th Int. Conf. Softw. Eng.*, 1985, pp. 268–272.
[2] D. E. Perry, N. A. Staudenmayer, and L. G. Votta, "People, organizations, and process improvement," *IEEE Softw.*, vol. 11, no. 4, pp. 36–45, Jul. 1994.
[3] J. Singer, T. Lethbridge, N. Vinson, and N. Anquetil, "An examination of software engineering work practices," in *Proc. CASCON 1st Decade High Impact Papers*, 2010, pp. 174–188.
[4] B. W. Boehm, "Improving software productivity," *IEEE Comput.*, vol. 20, no. 9, pp. 43–57, Sep. 1987.
[5] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann, "Software developers' perceptions of productivity," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2014, pp. 19–29.
[6] R. van Solingen, E. Berghout, and F. van Latum, "Interrupts: Just a minute never is," *IEEE Softw.*, vol. 15, no. 5, pp. 97–103, Sep./Oct. 1998.
[7] S. T. Iqbal and E. Horvitz, "Disruption and recovery of computing tasks: Field study, analysis, and directions," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2007, pp. 677–686.
[8] M. Czerwinski, E. Horvitz, and S. Wilhite, "A diary study of task switching and interruptions," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2004, pp. 175–182.
[9] C. Parnin and S. Rugaber, "Resumption strategies for interrupted programming tasks," *Softw. Qual. J.*, vol. 19, no. 1, pp. 5–34, 2011.
[10] V. M. González and G. Mark, "Constant, constant, multi-tasking craziness: Managing multiple working spheres," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2004, pp. 113–120.
[11] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *Proc. 29th Int. Conf. Softw. Eng.*, 2007, pp. 344–353.

[12] R. Minelli, A. Mocci, and M. Lanza, "I know what you did last summer—an investigation of how developers spend their time," *Proc. 23rd IEEE Int. Conf. Program Comprehension*, 2015, pp. 25–35.

[13] S. Amann, S. Proksch, S. Nadi, and M. Mezini, "A study of visual studio usage in practice," in *Proc. 23rd IEEE Int. Conf. Softw. Anal. Evol. Reengineering*, 2016, pp. 124–134.

[14] RescueTime. (2017).[Online]. Available: rescuetime.com, retrieved Jan. 16, 2017.

[15] Codealike. (2017). [Online]. Available: codealike.com, retrieved Jan. 16, 2017.

[16] M. Zhou and A. Mockus, "Developer fluency: Achieving true mastery in software projects," in *Proc. 18th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2010, pp. 137–146.

[17] J. Blackburn, G. Scudder, and L. van Wassenhove, "Improving speed and productivity of software development: A global survey of software developers," *IEEE Trans. Softw. Eng.*, vol. 22, no. 12, pp. 875–885, Dec. 1996.

[18] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity," in *Proc. 2nd ACM-IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2008, pp. 2–11.

[19] W. S. Humphrey, "Using a defined and measured personal software process," *IEEE Softw.*, vol. 13, no. 3, pp. 77–88, May 1996.

[20] P. M. Johnson, et al., "Beyond the personal software process: Metrics collection and analysis for the differently disciplined," in *Proc. 25th Int. Conf. Softw. Eng.*, 2003, pp. 641–646.

[21] B. Vasilescu, et al., "The sky is not the limit: Multitasking across GitHub projects," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 994–1005.

[22] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: A study of developer work habits," in *Proc. 28th Int. Conf. Softw. Eng.*, 2006, pp. 492–501.

[23] M. K. Gonçalves, L. R. de Souza, and V. M. González, "Collaboration, information seeking and communication: An observational study of software developers' work practices," *J. Universal Comput. Sci.*, vol. 17, no. 14, pp. 1913–1930, 2011.

[24] T. Roehm, R. Tiarks, R. Koschke, and W. Maalej, "How do professional developers comprehend software?" in *Proc. 34th Int. Conf. Softw. Eng.*, 2012, pp. 255–265.

[25] E. Cutrell, M. Czerwinksi, and E. Horvitz, "Notification, disruption, and memory: Effects of messaging interruptions on memory and performance," in *Proc. Conf. Human-Comput. Interaction*, 2000, pp. 263–269.

[26] J. Chong and R. Siino, "Interruptions on software teams: A comparison of paired and solo programmers," in *Proc. 20th Anniversary Conf. Comput. Supported Cooperative Work*, 2006, pp. 29–38.

[27] L. A. Perlow, "The time famine: Toward a sociology of work time," *Administ. Sci. Quart.*, vol. 44, no. 1, pp. 57–81, 1999.

[28] C. Parnin and R. DeLine, "Evaluating cues for resuming interrupted programming tasks," *Proc. 28th Int. Conf. Human Factors Comput. Syst.*, 2010, Art. no. 93.

[29] G. Mark, V. M. Gonzalez, and J. Harris, "No task left behind?: Examining the nature of fragmented work," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2005, pp. 321–330.

[30] B. P. Bailey, J. A. Konstan, and J. V. Carlis, "The effects of interruptions on task performance, annoyance, and anxiety in the user interface," in *Proc. INTERACT*, 2001, vol. 1, pp. 593–601.

[31] M. Czerwinski, E. Cutrell, and E. Horvitz, "Instant messaging: Effects of relevance and timing," in *Proc. Conf. Human-Comput. Interaction People Computers*, 2000, vol. 2, pp. 71–76.

[32] G. Mark, D. Gudith, and U. Klocke, "The cost of interrupted work: More speed and stress," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2008, pp. 107–110.

[33] H. Sanchez, R. Robbes, and V. M. Gonzalez, "An empirical study of work fragmentation in software evolution tasks," in *Proc. IEEE 22nd Int. Conf. Softw. Anal. Evol. Reengineering*, 2015, pp. 251–260.

[34] S. Consolvo, et al., "Activity sensing in the wild: A field trial of UbiFit garden," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2008, pp. 1797–1806.

[35] D. M. Bravata, et al., "Using pedometers to increase physical activity and improve health: A systematic review," *J. Amer. Med. Assoc.*, vol. 298, no. 19, pp. 2296–2304, 2007.

[36] B. J. Fogg, *Persuasive Technology: Using Computers to Change What We Think and Do*. Amsterdam, The Netherlands: Elsevier, 2003.

[37] I. Li, A. Dey, and J. Forlizzi, "Understanding my data, myself: Supporting self-reflection with Ubicomp technologies," in *Proc. 13th Int. Conf. Ubiquitous Comput.*, 2011, Art. no. 405.

[38] V. Hollis, A. Konrad, and S. Whittaker, "Change of heart: Emotion tracking to promote behavior change," *Proc. 33rd Annu. ACM Conf. Human Factors Comput. Syst.*, 2015, pp. 2643–2652.

[39] T. Fritz, E. M. Huang, G. C. Murphy, and T. Zimmermann, "Persuasive technology in the real world: A study of long-term use of activity sensing devices for fitness," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2014, pp. 487–496.

[40] Y.-H. Kim, J. H. Jeon, E. K. Choe, B. Lee, K. Kim, and J. Seo, "TimeAware: Leveraging framing effects to enhance personal productivity," in *Proc. CHI Conf. Human Factors Comput. Syst.*, 2016, pp. 272–283.

[41] Hubstaff. (2017). [Online]. Available: http://hubstaff.com, retrieved Jan. 16, 2017.

[42] Wakatime. (2017). [Online]. Available: wakatime.com, retrieved Jan. 16, 2017.

[43] C. E. Walston and C. P. Felix, "A method of programming measurement and estimation," *IBM Syst. J.*, vol. 16, no. 1, pp. 54–73, 1977.

[44] P. Devanbu, S. Karstu, W. Melo, and W. Thomas, "Analytical and empirical evaluation of software reuse metrics," in *Proc. 18th Int. Conf. Softw. Eng.*, 1996, pp. 189–199.

[45] V. Nguyen, L. Huang, and B. Boehm, "An analysis of trends in productivity and cost drivers over years," in *Proc. 7th Int. Conf. Predictive Models Softw. Eng.*, 2011, pp. 3:1–3:10.

[46] A. J. Albrecht, "Measuring application development productivity," in *Proc. IBO Conf. Appl. Develop.*, 1979, pp. 83–92.

[47] C. Jones, "Software metrics: Good, bad and missing," *IEEE Comput.*, vol. 27, no. 9, pp. 98–100, Sep. 1994.

[48] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," *ACM Trans. Softw. Eng. Methodology*, vol. 11, no. 3, pp. 309–346, 2002.

[49] C. Treude, F. F. Filho, and U. Kulesza, "Summarizing and measuring development activity," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, 2015, pp. 625–636.

[50] W. S. Humphrey, *Introduction to the Personal Software Process*, 1st ed. Reading, MA, USA: Addison-Wesley, 1996.

[51] S. Wagner and M. Ruhe, "A systematic review of productivity factors in software development," in *Proc. 2nd Int. Workshop Softw. Productiv. Anal. Cost Estimation*, 2008, pp. 1–6.

[52] D. Graziotin, X. Wang, and P. Abrahamsson, "Are happy developers more productive?" in *Product-Focused Software Process Improvement*. Berlin, Germany: Springer, 2013, pp. 50–64.

[53] I. A. Khan, W.-P. Brinkman, and R. M. Hierons, "Do moods affect programmers' debug performance?" *Cogn. Technol. Work*, vol. 13, no. 4, pp. 245–258, 2011.

[54] A. Mathur, M. V. D. Broeck, G. Vanderhulst, A. Mashhadi, and F. Kawsar, "Tiny habits in the giant enterprise: Understanding the dynamics of a quantified workplace," in *Proc. Joint Int. Conf. Pervasive Ubiquitous Comput. Int. Symp. Wearable Comput.*, 2015, pp. 577–588.

[55] G. Mark, S. T. Iqbal, M. Czerwinski, P. Johns, and A. Sano, "Email duration, batching and self-interruption: Patterns of email use on productivity and stress," in *Proc. CHI Conf. Human Factors Comput. Syst.*, vol. 21, no. 1, 2016, pp. 98–109.

[56] G. Mark, S. T. Iqbal, M. Czerwinski, P. Johns, and A. Sano, "Neurotics can't focus: An in situ study of online multitasking in the workplace," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2016, pp. 1739–1744.

[57] C. A. Monk, J. G. Trafton, and D. A. Boehm-Davis, "The effect of interruption duration and demand on resuming suspended goals," *J. Exp. Psychology: Appl.*, vol. 14, no. 4, pp. 299–313, 2008.

[58] D. A. Epstein, M. Caraway, C. Johnston, A. Ping, J. Fogarty, and S. A. Munson, "Beyond Abandonment to next steps: Understanding and designing for life after personal informatics tool use," in *Proc. CHI Conf. Human Factors Comput. Syst.*, 2016, pp. 1109–1113.

[59] G. Mark, S. T. Iqbal, M. Czerwinski, and P. Johns, "Bored mondays and focused afternoons: The rhythm of attention and online activity in the workplace," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2014, pp. 3025–3034.

[60] (2017). [Online]. Available: ifi.uzh.ch/seal/people/meyer/productive-workday

[61] StackOverflow, "Stack overflow developer survey 2015, " (2016). [Online]. Available: http://stackoverflow.com/research/developer-survey-2015, retrieved Jan. 16, 2017.

[62] J. Taillard, P. Philip, and B. Bioulac, "Morningness/eveningness and the need for sleep," *J. Sleep Res.*, vol. 8, no. 4, pp. 291–295, 1999.

[63] M. Züger and T. Fritz, "Interruptibility of software developers and its prediction using psycho-physiological sensors," in *Proc. 33rd Annu. ACM Conf. Human Factors Comput. Syst.*, 2015, pp. 2981–2990.
[64] PomodoroTechnique. (2017). [Online]. Available: pomodoro technique.com, retrieved Jan. 16, 2017.

André N. Meyer is working toward the PhD degree in computer science under the supervision of T. Fritz at the University of Zurich and is also working for the information technology industry as an application developer (and intern). In his research, he focuses on developers' productivity and on building tools for developers' awareness of their work. His homepage is http://www.andre-meyer.ch.

Laura E. Barton is working toward the undergraduate degree in computer science at the University of British Columbia, while interning as a software engineer and exploring research opportunities in the field.

Gail C. Murphy received the BSc degree in computing science from the University of Alberta and the MS and PhD degrees in computer science and engineering from the University of Washington. She is a professor in computer science and associate dean (Research and Graduate Programs) in the faculty of science with the University of British Columbia. She is also a co-founder and chief scientist with Tasktop Technologies, Inc. Her research interests include software developer productivity and software evolution. She is a fellow of the Royal Society of Canada and a member of the IEEE Computer Society.

Thomas Zimmermann received the PhD degree from Saarland University, Germany. He is a senior researcher with Microsoft Research. His research interests include software productivity, software analytics, and recommender systems. He is a member of the IEEE Computer Society. His homepage is http://thomas-zimmermann.com.

Thomas Fritz received the diploma degree from Ludwig-Maximilians-University Munich, Germany, in 2005 and the PhD degree from the University of British Columbia, Canada. He is an assistant professor in the Department of Informatics, University of Zurich, Switzerland. His research focuses on empirically studying software developers and on using personal and biometric data to improve developers' productivity. He is a member of the IEEE Computer Society and currently a member At-Large of IEEE TCSE.