Software Developers' Perceptions of Productivity

André N. Meyer, Thomas Fritz
University of Zurich
Zurich, Switzerland
andre.meyer@uzh.ch, fritz@ifi.uzh.ch

Gail C. Murphy
University of British Columbia
Vancouver, BC Canada
murphy@cs.ubc.ca

Thomas Zimmermann
Microsoft Research
Redmond, WA USA
tzimmer@microsoft.com

ABSTRACT

The better the software development community becomes at creating software, the more software the world seems to demand. Although there is a large body of research about measuring and investigating productivity from an organizational point of view, there is a paucity of research about how software developers, those at the front-line of software construction, think about, assess and try to improve their productivity. To investigate software developers' perceptions of software development productivity, we conducted two studies: a survey with 379 professional software developers to help elicit themes and an observational study with 11 professional software developers to investigate emergent themes in more detail. In both studies, we found that developers perceive their days as productive when they complete many or big tasks without significant interruptions or context switches. Yet, the observational data we collected shows our participants performed significant task and activity switching while still feeling productive. We analyze such apparent contradictions in our findings and use the analysis to propose ways to better support software developers in a retrospection and improvement of their productivity through the development of new tools and the sharing of best practices.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Human Factors

Keywords

Retrospection, productivity, goal setting

FSE'14 , November 16–22, 2014, Hong Kong, China.

1. INTRODUCTION

There is a common refrain that repeats itself in similar forms every few years: the inability for enough software to be produced to satisfy the needs of the world. In 1968, attendees at the first NATO software engineering conference coined the term software crisis [34]. In 1972, Dijkstra wrote about "programming [becoming a] gigantic problem" [16, p. 861]. In 1987, Boehm wrote about the growing demand for software [6]. In 1996, Gibbs wrote about the chronic software crisis [19]. And in 2011, Andreessen wrote about software eating the world, expressing that the need for software keeps outstripping the ability to produce the software [2].

There are a couple of ways of addressing the gap between software demand and supply. We could try to reduce the demand, namely the world's appetite for software. This approach seems unlikely to succeed. Or, we could try to increase the supply, namely our ability to produce software. In this paper, we consider one way to address supply: how we might improve the productivity of software developers.

A substantial amount of research into the meaning of software productivity has been undertaken over the past four decades (Section 2). Much of this research introduces particular definitions of productivity, such as computing productivity based on the number of source lines of code per hour [15]. Another body of research considers organizational issues associated with productivity, such as the effect of the workplace on programmer performance [14]. There is also research focused at specific tools and approaches for improving productivity. For example, the Personal Software Process (PSP) aims to help software developers improve their skills and quality of work by tracking a number of measures, including schedule data [22]. Surprisingly, there has been no work that we have been able to find that considers when software developers perceive themselves to be productive and when they perceive themselves as unproductive. This information can help inform how productivity is defined, measured, assessed and supported by tools and best practices.

In this paper, we gather data about software developers' perceptions of productive and unproductive work through two studies: a survey (Section 3) and an observational study (Section 4). The survey we conducted had 379 responses from individuals with an average of 9.2 (±7.3) years professional software development experience. Our analysis of the survey responses found, amongst other results, that developers think about productive days in terms of ones in which many or big tasks are completed without significant context switching or interruption.

The observational study we conducted involved observing 11 professional software developers from three companies at work for four hours each. As the developers worked, we collected detailed logs of the tasks worked on and the programs used to perform work, gathering a total of 2650 log entries. We defined a task as a piece of work with a specific goal, such as fixing a bug, and an activity as an action undertaken by the developer during his work, such as navigating code or reading an email.

We also performed semi-structured follow-up interviews of the participants in this study. From this data, we found that significant context switching between tasks and activities can occur with developers still perceiving themselves as productive. The number of task switches we observed, 13.3 (±8.5) per hour on average, is similar to other reports in the literature [20, 29]. The number of activity switches is larger than reported in previous studies at 47 (±19.8) times per hour on average. From this study, we gained insights into the complexities involved with helping developers assess their own productivity.

We conclude the paper with a discussion of ways in which we might help developers retrospect upon their productivity and share best practices (Section 5).

This paper makes three contributions:

- it presents results from a survey of 379 professional software developers, describing their perceptions of when their work is productive and what measurements of productivity they might find useful,
- it presents the results of an observational study of 11 professional software developers, showing that productive work can occur in the presence of some kinds of fast context switches, and
- it synthesizes the results of these two studies to suggest ways in which we might better support developers in reflecting upon their productivity and sharing best practices.

2. RELATED WORK

Definitions of productivity share characteristics of typically being about efficiency, inputs and outputs. As one example, the Oxford Dictionary defines productivity as "effectiveness of productive effort, especially in industry, as measured in terms of the rate of output per unit of input" [37]. Often, the unit of input is time-based. Most research in software engineering defines productivity along similar lines; here are some examples:

- number of modification requests and *added* lines of code per year [32],
- number of tasks per month [41],
- number of function points per month [25],
- number of source lines of code per hour [15],
- number of lines of code per person month of coding effort [5],
- amount of work completed per reported hour of effort for each technology [28],
- ratio of produced logical code lines and spent effort [21],
- average number of logical source statements output per month over the product development cycle [30],
- total equivalent lines of code per person-month [35],
- resolution time defined as the time, in days, it took to resolve a particular modification request [9], and
- number of editing events to number of selection and navigation events needed to find where to edit code [27].

In this paper, we do not attempt to define productivity for developers but instead we investigate how developers themselves think about productive versus non-productive work.

In contrast, the majority of the work we could find about software development productivity focused on the organizational, not the personal, level. DeMarco and Lister found evidence that characteristics of the workplace and organization have significant influence on the performance of programmers [14]. Boehm looked at large-scale possibilities for improving outputs, such as hiring well and using better languages and tools [6]. Blackburn et al. correlated lines of code per person month to aspects such as program size and team size from survey data collected from Western Europe, Japan and US, finding that time-to-market correlated with higher productivity, while larger teams tend to have lower productivity [5]. Through a systematic literature review, Wagner and Ruhe distilled a list of the main factors influencing productivity, separating them into technical (e.g., product complexity) and soft (e.g., manager capability) factors [40].

There are a lot fewer studies on productivity at the level of an individual developer where the focus is on the study of productivity, not on the effect of introducing a new tool or process to improve productivity. One of the few recent studies is by Kamma and Jalote who recorded and analyzed the screens of highly productive developers (as identified by managers) to identify characteristics of what makes developers productive when performing model-based testing activities. They found differences in the behaviors of the highly productive developers that could be captured as best practices, such as that high productivity programmers moved all required information to a common place and avoided referring to multiple documents when writing test cases [26]. The studies we report on in this paper help describe more general work practices that developers themselves see as productive.

To try to improve the productivity of software developers, many approaches have been suggested (e.g., Extreme Programming [3]). A few approaches have been aimed more specifically at improving productivity. The most notable of these is the Personal Software Process (PSP), which aims to help individuals improve their skills and quality of work by collecting (often manually) a set of basic PSP measures such as time, size, quality (defects), and schedule data [23, 22]. Johnson et al. [24] observed a "PSP adoption problem," which they attributed to the high overhead of PSP-style metrics collection and analysis, and the requirement that PSP users switch between product development and process recording. To eliminate overhead and context switching, they introduced Hackystat which fully automates both data collection and analysis. Our work is orthogonal to PSP and Hackystat in that we explore what it means to developers to be productive and how they perceive productivity. This information can inform the design of new tools focused on aspects of productivity that developers care about the most.

In the last twenty years, an increasing number of studies were conducted about how developers work. Perry and colleagues' paper that collected time diaries from developers and performed direct observation of developers at work to see where they spent their time was one of the first [36].

Table 1: Sample Survey Questions.

Q10	Are you satisfied with your productivity last week? (very unsatisfied, unsatisfied, undecided, satisfied, very satisfied)
Q11	How did you assess if you were productive last week?

They found that work was typically performed in two hour chunks, progress on a particular development task could end up being blocked for various reasons, over half of the developers' time was spent in interactive activities other than coding and there were seven unique personal contacts per day on average. The data we present in this paper on the observational study we performed also found situations in which developers were blocked (also similar to Ko et. al. [29]), but we saw much less time being spent on any one task. Other observational based studies have focused on specific areas of development, such as novice programming [4], program comprehension [38], software dependencies [13], information needs [29], and change tasks [39]. The work we report on in this paper is unique in considering the condition under which developers perceive they are productive.

3. STUDY 1: SURVEY

To gain a broad sense of what productivity means to software developers and how developers assess their productivity, we conducted a survey.

3.1 Participants and Method

The online survey had 28 questions: 8 on the participants' background and experience, 7 on perceptions of productivity, 7 on goal setting and monitoring, 3 on techniques for improving and monitoring productivity, and 3 on the raffle participation and dissemination of results. 16 of the 28 questions had a closed set of answers from which a participant selected, while 12 of the questions were open-ended. Table 1 shows an example of a closed and an open-ended question. None of the questions were obligatory and a participant was allowed to drop out at any time. Depending on a participant's answers, some questions were filtered and not presented to avoid asking unnecessary questions. Interested readers can download the complete survey from our web site [1].

We announced the online survey on Twitter and in several big online developer forums, including two big German-speaking IT news portals.[1] We also advertised within Microsoft, sending a personalized email to 1500 Microsoft employees and directing them to a special internal posting of the survey. To incentivize participants, we held a raffle for the online participants to win two 200 US$ Amazon gift certificates and one for the Microsoft participants to win two 50 US$ Amazon gift certificates. We ended up with 379 valid responses: 185 from the general advertisements and 194 from Microsoft. Of the 185 participants that completed the survey from the general advertisement, the majority of the participants was from Germany (32%) and Switzerland (16%). Of all 379 participants, more than 93.4% of the participants listed their job as software developer; the remaining 6.6% reported having experience in software development. The average professional software development experience per participant was 9.2 (±7.3) years.

[1] `heise.de/developer` and `pocketpc.ch`, verified 03/15/14

Table 2: Top 5 Reasons for a Productive Workday. (#, %: Number and Percentage of Participants)

I have a productive workday when I . . .	#	%
complete tasks or goals	192	53.2
have no/few interruptions and distractions	182	50.4
have no meetings	79	21.9
have clear goals and/or requirements are set	72	19.9
plan my workday	62	17.2

3.2 Results

We begin by describing when and in what conditions participants perceived their software development activities as productive. We then describe how participants think they could assess or measure their productivity. Since the responses to the open-ended questions largely overlap and there are only minor differences between the ratings by participants from within Microsoft and from the general advertisement, we present the results aggregated over all participants.

A Productive Day.

We asked each participant to complete the sentence "I have a productive workday when . . . " in up to three different ways. Table 2 summarizes the top five reasons mentioned by participants for having a productive workday. Most participants responded that their workday is productive when they complete tasks, achieve their planned goals or make progress on their goals (stated by 192, 53.2%). The second most mentioned reason for having a productive workday is getting into a "flow" without many "context-switches" and with no or few interruptions and distractions (182, 50.4%). Different kinds of distractions were described, ranging from interruptions from co-workers asking questions, phone calls, infrastructure issues, such as waiting for a build to complete, to disrupting background-noise in the office. The remaining three reasons of the top five all had substantially lower support: having no meetings (79, 21.9%), having clear goals (72, 19.9%) and planning of a workday (62, 17.2%).

Productive and Unproductive Activities.

Other questions in the survey focused on developers' perceptions of productivity on a finer-grained level. For example, we asked participants in two open-ended questions about the activities they consider particularly productive or unproductive in a workday.

The top five productive and unproductive activities mentioned by participants are summarized in Table 3. Unsurprisingly, the activities mentioned most frequently as particularly productive were coding related, including coding, testing, bug fixing and code reviews (236 participants, 71.5%). Despite this strong support for coding activities as productive, some participants (47, 14.2%) did mention certain aspects of coding as unproductive; these participants mostly differentiated between productive coding activities, such as implementing a feature, and unproductive ones, mainly debugging and testing.

Many other activities had mixed responses as to whether they were productive or unproductive. Most participants (191, 57.9%) think meetings are unproductive and a waste of time. However, 57 (17.3%) participants considered formal and informal meetings as productive, particularly when the

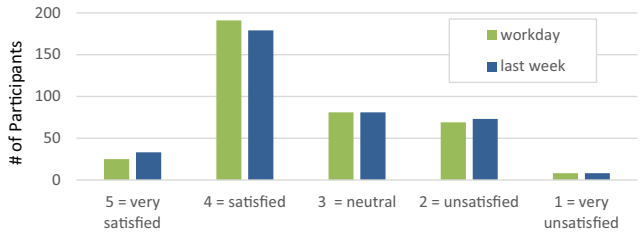

Figure 1: Developers' Productivity Satisfaction.

meetings include decision making, have a clear focus and goals, improve relationships between developers, help others and when all meeting attendees are well-prepared. The main reasons stated for unproductive meetings were missing goals, lacking a preparation, too many participants attending the meeting, or going over time.

Participants were also mixed on their views of email. 62 (18.8%) participants considered time spent on email as unproductive, while a few (10, 3%) stated reading and writing emails as a productive activity. The amount of emails appears to be a reason contributing to this activity being seen as unproductive:

Tracking and responding to tons of emails [and] email communications going back and forth for days together with no closure. (MS23)

Other activities mentioned as being productive included planning (25, 7.6%), writing documentation and (administrative) reports (22, 6.7%), modeling or designing an architecture (18, 6.7%), or learning new things (11, 3.3%).

Assessing Productivity.

When asked to rate their productivity on the previous workday and workweek on a Likert-scale from 1 (very unsatisfied) to 5 (very satisfied), most participants were satisfied with their work (see Figure 1, median of 4, mean of 3.42, ±0.94). Only a few participants mentioned being very satisfied (7.7%) or very unsatisfied (2.1%). The participants reported assessing their productivity in a variety of ways. Most (242, 78.1%) assessed their productivity on their progress in the past workday or workweek, largely through tasks and work items completed. Several participants (86, 27.8%) mentioned other measures, such as lines of code, number of commits, number of bugs found or fixed, number of test cases written and code reviews completed and number of emails sent. Others (43, 13.9%) stated that they use their feelings to assess their productivity.

Table 3: Top 5 Productive and Unproductive Activities. (#, %: Number and Percentage of Participants)

	#	%
Productive activities		
coding (implementing new features, testing, bug fixing, code reviews)	236	71.5
meetings	57	17.3
planning	25	7.6
reading/writing documentation and reports	22	6.7
designing or modeling an architecture (i.e. solution for a programming problem)	18	5.5
Unproductive activities		
meetings	191	57.9
reading/writing emails	62	18.8
unplanned work (e.g., solving problems, fighting fires, unplanned tasks)	58	17.6
coding (testing, debugging, maintenance)	47	14.2
reading/writing documentation and reports	25	7.6

Table 4: What Participants Want to Measure. (#, %: Number and Percentage of Participants)

	#	%
Activities (how much time was spent where)	67	27.0
Achievements (actual work done, progress)	44	17.7
Value (usefulness of completed work, success of feature, value to customer)	41	16.5
Time per task ratio	39	15.7
Number of context switches and distractions	36	14.5

Measuring Productivity.

We also asked participants about which measures might be helpful to them to assess their productivity. In particular, we asked participants to rate 23 possible measures on a five point Likert-scale, with 1 - they strongly disagree that the metric would help them to assess their personal productivity and 5 - they strongly agree. The 23 measures were identified by the authors in an iterative process taking into account related work and responses from our pilot survey participants. The results are presented in Figure 2.

The metric with the highest rating is "The number of work items (tasks, bugs) I closed." with a mean of 3.88 (±1.22). This supports our findings presented above that participants assess their productivity often based on their tasks or work items completed. Overall, the means of all metrics lie between 2 and 4 with an overall mean of 3.14 (±0.35). 16 metrics were rated 3 or higher while only 7 metrics have values slightly below 3, showing that while the number of work items closed might be considered helpful by a wide range of participants, there is no single metric that is considerably better than others to assess a developer's productivity, similar to what Card states in [8]. Furthermore, participants usually rated several metrics as helpful, and differed in which metrics they considered more helpful, suggesting that measuring one's productivity is an individual task that varies across developers. When further asked in an open-ended question on what one would want to measure to meet ones goals and improve ones productivity, 221 (89.1%) of the participants who answered the question mentioned one or multiple measurements. The measures varied a lot across participants. The top five mentioned pieces of information participants wanted to measure are presented in Table 4. 67 (27.0%) participants are interested in better knowing how they spend their time on their computer, in meetings, or taking breaks, as well as in comparing their time spent coding with time spent in meetings, time spent on personal tasks and total work time. Developers were also interested in the value of their work. 41 (16.5%) participants mentioned that performing useful, necessary, and interesting work and having the feeling of being necessary to the team or product is very important.

27 (10.9%) participants explicitly stated that they would not want to be measured, as they either think measuring decreases their productivity, have privacy concerns or think it is not possible to accurately measure productivity.

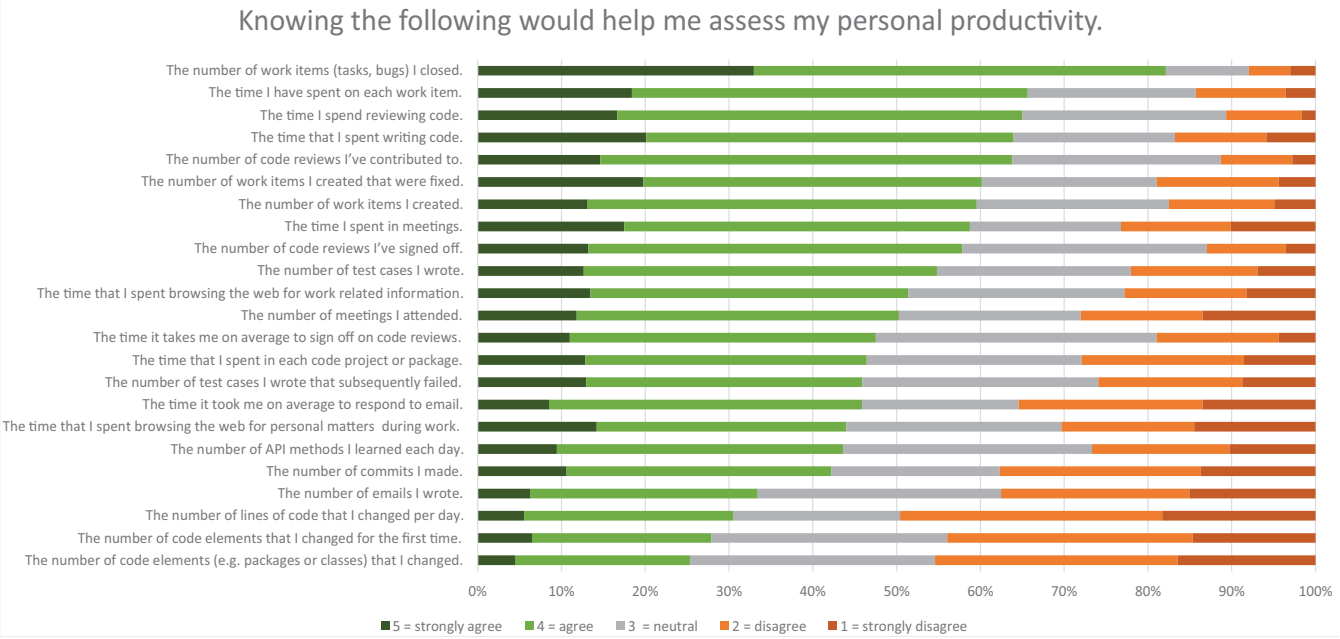

Figure 2: Metrics for Assessing Productivity.

3.3 Threats to Validity

Since the survey participants might not be representative of the general population of software developers, the generalizability of our survey results might be limited. To mitigate this risk, we advertised our survey through various channels to a large audience. Having gathered data from software developers from various countries, and with different levels of open-source and closed source experience, we believe that our sample is fairly representative of software developers and that it provides an interesting perspective. Participants could freely decide whether to participate in the study or not (self-selection). They were informed about the survey's topics, an estimated duration for the participation and offered a raffle to incent their participation. This could have biased the selection of participants as only participants who could spare enough time or were interested in the incentive might have participated. We tried to mitigate this risk by advertising through various channels and offering a very generic incentive.

4. STUDY 2: OBSERVATION

The survey results raise many questions. For instance, is there a common meaning between developers for what constitutes a task or a context switch? Are all interruptions necessary? How, when and what kind of email must developers process? To investigate some of the survey results in more depth, we conducted an observational study of software developers at work including follow-up interviews. To focus the observational study, we extracted three themes from the survey results:

1. the role of tasks in how developers work and how developers view productivity,
2. different views on the effect on productivity of various kinds of activities, and
3. the role of uninterrupted work to be in "the flow".

4.1 Participants and Method

This observational study involved 11 professional software developers who were recruited from three international software development companies of varying size (Table 5). We used personal contacts and a recruiting email to solicit participation. Of the 11 participants, two were female and nine were male, and all resided either in the US or Canada. Participants had an average of 5.4 years (±3.4, ranging from 0.5 to 8 years) of professional software development experience and an average of 13.3 years (±8.3, ranging from 1.5 to 29 years) of total development experience. The primary work area of all participants was development and the roles varied between individual contributor and lead.

The study had two parts. In the first part, we observed each participant for a total of four hours on a single workday: two hours before and two hours after lunch. In four

Table 5: Observational Study Participants (Dev: Developm., PM: Project Management, IC: Individ. Contributor).

ID	Primary Work Area	Role	Dev Experience Prof.	Dev Experience Overall
Company R				
R1	Dev	IC	8.0	14.0
R2	Dev	Lead	7.5	8.5
R3	Dev	Lead	5.0	11.0
R4	Dev	IC	8.0	13.0
Company S				
S1	Dev	IC/Lead	7.5	14.5
S2	PM/Dev	IC/Lead	4.0	22.0
S3	Dev	IC	0.5	1.5
S4	Dev	IC	12.0	29.0
Company T				
T1	Dev	IC	2.0	22.0
T2	Dev	IC	2.0	5.0
T3	Dev	IC	3.0	6.0

cases, we had to adapt the two sessions to accommodate for the participant's availability, still ensuring to have at least one hour in the morning or afternoon and all four hours on the same day. Before the first session, the observer, one of the authors of this paper, introduced himself to the participant as well as any colleagues working nearby, asking for all to ignore him as much as possible. He then provided a short introduction to the study and placed himself behind the participant to prevent distractions while still being able to read the screen contents of the participant, before asking the participant to continue his workday. The observer noted in an observation log each time the participant switched a program on his computer, was interrupted by others or interrupted him or herself, or switched a task while being in the same program. Each entry in the observation log consists of a time stamp, a reason for the switch, the program that the participant switched to and the task that the participant was working on. We defined a task as a piece of work with a specific goal or intention, such as fixing a bug or reviewing code changes for a specific work item. We inferred tasks from the active programs on the screen and the information considered in those programs, such as a work item consulted before beginning to code. Ten minutes into the first session, the observer briefly interrupted the participant to validate the tasks the participant was working on to ensure the quality of the collected transcripts. The preparation and process of the observation was inspired by Mintzberg's protocol of a structured observation session [31].

All observations were conducted by the same observer. For the first session of the first participant, we cross-checked the observations by having a second observer also record observations. Almost all events were noted by both observers. Of the total of 202 entries for the first session, less than 10 log entries (4.9%) were not contained in both observer's logs, while all others matched, suggesting a high accuracy of the primary observer's logs. Due to confidentiality reasons, we are not able to share the observation logs.

In the second part of the study, following the observations, we interviewed each participant using a semi-structured approach by using a set of prepared questions as general guidance. As part of this interview, the observer briefly discussed the tasks a participant worked on during the two observation sessions to verify the accuracy of the observation notes. The bulk of the interview focused on gaining further insights into the observations and to investigate themes arising from the survey results. In particular, the questions focused on participants' perception and reflection of productivity, context switches, work items, code check-ins, meetings and emails and whether and how information on these might help to assess a developer's productivity. Interviews took between 34 and 51 minutes per participant, and were conducted in person. Interested readers can download the interview guidance notes from our website [1].

From the observation sessions, we collected transcripts of a total of 44 hours of work with a total of 2650 observation events over all 11 developers. During the data analysis, we further determined the number of tasks a developer worked on and categorized each program into one of the activity categories listed in Table 6, which we inferred from the survey results and the observations using an open coding technique. The logged time stamps were then used to calculate the duration spent on interruptions, activities and tasks, amongst others.

Table 6: Activity Categories for Observations.

Category		% of time over all Observ.
Development		
Code	reading/editing/navigating code	32.3%
Debug	debugging	3.9%
VC	reading/accepting/submitting changes	2.4%
TestApp	testing application outside IDE	11.7%
Review	performing code reviews	2.3%
DevOther	other related to development	4.1%
Email	reading/writing emails	4.9%
Planning	editing work items/tasks/todos; creating/changing calendar entries	7.9%
ReadWriteDoc	reading/editing documents and other artifacts, e.g. pictures	2.7%
MeetPlanned	scheduled meeting/call	4.9%
MeetInformal	ad-hoc, informal communication; e.g. unscheduled phone call / IM, or colleague asks a question	13.1%
BrowsingRel	Internet browsing related to code/work/task	4.0%
BrowsingUnrel	Internet browsing work unrelated	0.4%
Other	Anything else, such as break or changing music	5.4%

4.2 Results

Figure 3 illustrates the work of each participant over the first observation session. Each participant is represented by a row. Each row is divided into segments, with each segment representing a particular task: when tasks are revisited the same grey-scale is used to represent the task. Figure 3 also shows the activities undertaken by each participant. The occurrence of the start of an activity is indicated with a particular colored shape. The lines emanating from the activity indicate the duration of an activity. For two participants, three hours of work are represented as these participants required the observation sessions to be split into one three hour session and one one hour session. Full graphs of both sessions are available here [1]. Our analysis of the data in this section considers all of the data collected over the four hours for each participant.

Theme 1: Tasks.

During the observation sessions, participants worked on between 2 and 10 tasks (mean of 4.8, ±2.3) each, such as fixing a bug, reviewing code, helping co-workers with a problem, or reading and writing emails. Each participant switched frequently between tasks with a mean task switch rate of 13.3 (±8.5) times per hour. The average time spent on each task was 6.2 (±3.3) minutes. When asked about whether the workday observed was productive or not, 8 of the 11 participants (73%) stated that they were fairly or very productive during the observation sessions. The number of tasks worked on in the time observed (2 to 10) is similar in magnitude to the number of working spheres (13), a concept similar to our use of task, observed by Gonzales and Marks in a study that included a small number of software developers [20]. The rate of task switching is similar to that reported in both [20] and [29].

The participants mentioned several reasons during the interview for why task switches occurred, including the need to help co-workers make progress (T3, from company T),

to unblock them (S3 and S4, from company S), and to interrupt themselves (all but two participants), similar to [29] and [12]. The participants also mentioned that task switches occurred when they were blocked themselves or waiting, such as waiting for a build to finish. In these cases, the participants mentioned that switching to email, code reviews, or other small tasks can help increase their productivity. Five of 11 participants (45%) also stated that making progress on either many tasks or a difficult one (S1, S2, T2, R2, R4) makes them perceive the day as productive. More experimentation is needed to further narrow down what "many" tasks means.

The results of this study also raise questions about the relationships between tasks and trackable work items. From the survey results, the highest rated measure for assessing productivity (Figure 2) was the number of work items closed. From our observations, we found that the participants worked on many more tasks than trackable work items. When the participants were asked during the interview how many work items they worked on during that day, answers ranged from one to 13 and all participants stated that the size of the work items varies.

There could be days where you've been working on one massive bug or issue or identified other sub-issues into it that need to be resolved, but there could be other days where you just got 20 bugs, but they are all little things. (R4)

The varying size of work items likely introduces problems for using the measure of the number of work items closed as a sole means of productivity assessment, at least for shorter time periods, such as a day. Participants did mention the use of the number of closed work items as one that is used to reflect on a team's productivity, such as in meetings associated with the end of an agile sprint.

Theme 2: Activities.

During the sessions, participants switched activities 47.0 (±19.8) times per hour, spending on average 1.6 (±0.8) minutes on an activity before switching. Table 6 shows the percentage of time across the observation periods that developers spent in each activity. Unsurprisingly, given our selection of participants, the largest amount of time was spent on coding (32.3%), with testing the application also taking an average of 11.7% of the observed time for each participant.

Based on an n-gram analysis of the activities recorded, we found that developers most frequently switched back and forth between coding and testing. Following coding the next activity was testing in 28.5% of the cases; a similar result holds for switches from testing to coding. Coding was also often superseded by an informal meeting (14.9%) either because someone asked a question or the participant asked someone else a question. After an informal meeting, developers either went to coding (26.1%), testing (14.8%) or right into planning (18.3%). Finally, emails were mostly checked while coding (18.8%) or testing (19.4%) and developers generally returned to testing (20.7%), coding (17.2%) or planning (19.5%) after the email check.

Interestingly, the second highest activity in terms of amount of time spent (Table 6) was informal meetings (13.1%), ranging from instant messages to a colleague interrupting and asking a question. During the interviews, participants described having between one and ten informal meetings per day, taking anywhere from one minute to two hours. The participants in this study agreed that unplanned, informal meetings are usually productive as they are generally short and efficient, and often succeed in helping to unblock a team member.

Unlike the survey respondents, participants in the observational study found all meetings generally productive, whether informal or formal. Participants in this study described having five to ten planned meetings in the past workweek, taking anywhere from 15 to 60 minutes each and uniformly stated that having more than two meetings a day decreases their productivity. Participants did describe formal meetings as more productive when only a few people are involved, there is a concrete outcome and the participant feels useful in the meeting.

Most survey respondents described email activities as unproductive (62, 18.8%) compared to productive (10, 3.0%). When interviewed, participants in the observational study did not consider emails as unproductive, given the small fraction of time that email took up in their day. Table 6 shows that the participants spent on average just under 5% of their time handling email during our observation. This amount of time is similar to the time spent in formal meetings and in taking breaks of various forms.

Each participant used on average 14.8 (±3.9) different programs during the observation. Several developers used multiple different programs to achieve a similar activity. For instance, a few participants used two different development environments to develop software or two different email clients to handle email.

Theme 3: Work Flow.

We were surprised by the number of task and activity switches performed by the developers we observed. In particular, because the survey respondents as well as observed developers had expressed concern that context switches lead to a loss of productivity while 8 of the 11 (72.7%) developers observed making so many switches in the four hours, felt that their sessions were productive. To understand the relationships between task switches, activity switches and context switches, we asked the participants in the observational study to define a context switch. The participants predominantly described a context switch as a change in thinking, as in:

When I have to stop thinking about one thing and start thinking about something else. (T1)

Participants did go on to discuss the range of context switches from small ones, such as getting distracted due to background noise or switching between programs when working on the same task (R2, R3), to switching between a main task and small other tasks such as code reviews or writing an email (R4) to switching between two cognitively different tasks (R1). Although all participants agreed that context switches generally reduced productivity, the cost or harm is dependent upon the duration of the switch, the reason for the switch and the focus on the current task. The longer the switch the more expensive:

[To] stop and work on a different task is a more costly context switch than writing a quick email. (S1)

Similarly, the more a developer is focused on a task, the more expensive the switch:

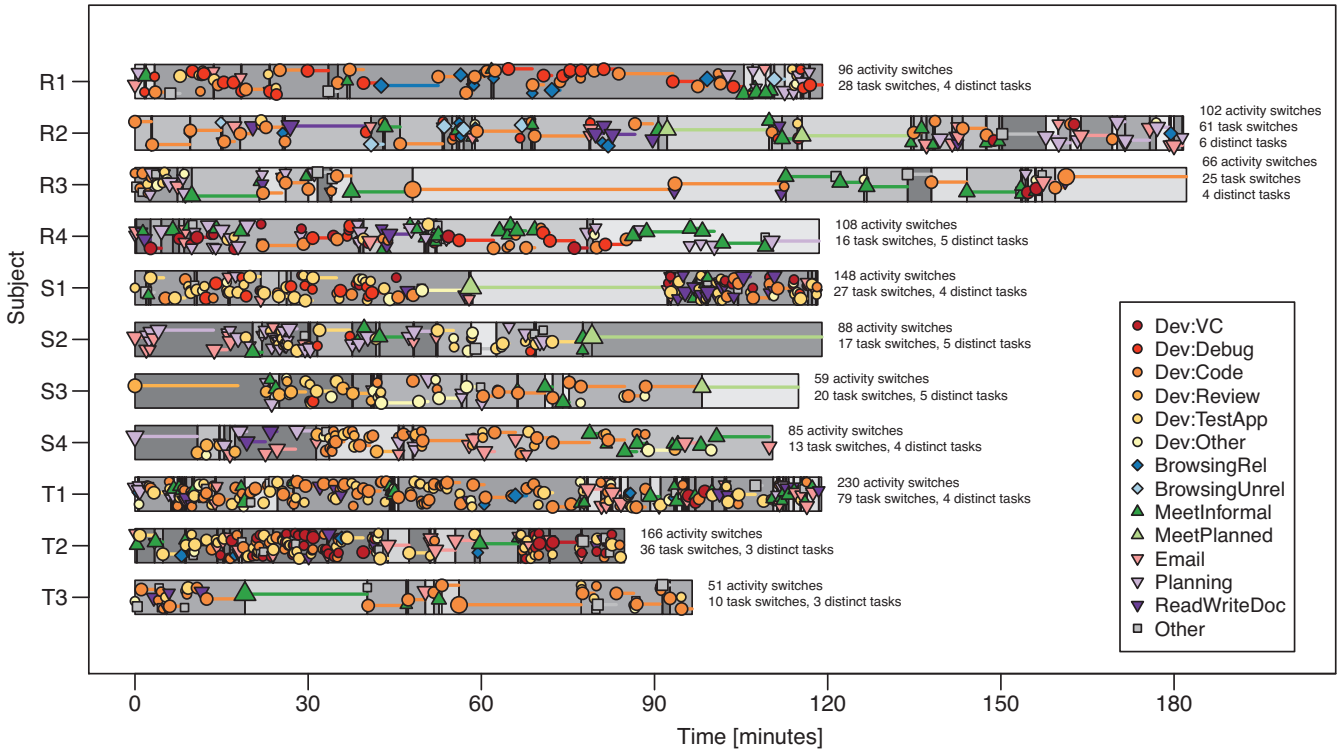

Figure 3: Developers' Activity and Task Switches in the First Observation Session (for a figure of *all* sessions, please visit `www.ifi.uzh.ch/seal/people/meyer/developers-productivity`). © Meyer, Fritz, Murphy, Zimmermann.

Depends on where I was, if it was a critical section, it is really hard to get back to focus on that task, even if it only was for like 30 seconds. (R3)

And also, switching away from a creative task for which more focus is required is more expensive than switching from a routine task (S2). Finally, a self-inflicted context switch, such as going for a coffee break or writing a quick, task-unrelated email, is less expensive than externally imposed switches, such as interruptions by co-workers or an unplanned task (S2, T2).

The distinctions in the kinds of context switches help explain why the participants felt their work sessions were productive despite high numbers of task and activity switches. In particular, we observed the participants often switching to simple tasks, such as reviewing code for 30 seconds or answering an email, without apparent significant impact to a primary task, such as fixing a bug. Developers also adapt to waiting times, such as waiting for builds to complete, by performing low cost switches to tasks on which they could make progress. These short fast task switches are even faster than those reported by Gonzales and Marks in a study of business analysts, software developers and managers; they reported fast task switches as spending approximately two and a half minutes at a time on email [20].

4.3 Threats to Validity

The small number of participants in our observation and interview study, the use of personal contacts for inviting participants and the short sessions capturing a total of 44 hours of work might limit the generalizability of the results of this study. We tried to address this threat by selecting participants from three different international software companies. Furthermore, the strength of our mixed method approach allowed us to triangulate findings obtained through the survey study with the results from the observations and follow up interviews. Additionally, participants were observed in their normal real-world work and not during an experimental exercise.

Another limitation might be the study setting of observing participants for 4 hours in one day. Participants could have had a particularly productive or unproductive day, or other factors could have influenced them. Furthermore, by sitting behind the participant and observing him or her, the observer might have been a source of distraction, influenced the work style or prevented interruptions by co-workers. We tried to mitigate this risk by splitting up the session into two, two-hour sessions, sitting as far away from the participant as possible and telling co-workers beforehand about the study and that they should continue and interrupt as usual.

The collection and categorization of data poses another threat to validity, since it is not straightforward to identify task switches or not miss any switches. To mitigate these risks, all observations were done by the same researcher and task switches were confirmed with participants. A cross-validation was also done for the first session, showing significant agreement between the two observers.

Finally, the interviews and qualitative analysis was performed by one author using grounded theory techniques, such as open, axial and selective coding. To avoid observer bias, several parts of the survey, interview and observation were analyzed and open coded by at least one more author.

5. DISCUSSION

Our findings contribute new knowledge about how developers perceive their own productivity and how it relates to their software development work. Despite the general interest of our participants in assessing and improving their productivity, there is little tool support to help them and few best practices recorded. Opportunities exist to better support developers in managing and improving their work to achieve higher productivity levels.

Tools for Retrospective Analysis.

Self-monitoring can provide valuable insights into one's own behavior and reflection about self-monitored information can be used to change one's behavior (e.g., [33], [17]). In particular, activity tracking devices have been shown to be successful in promoting individuals to adopt a more active lifestyle (e.g., [11, 18]). There has been growing interest in this area with devices such as the Fitbit[2], which tracks such information as the number of steps per day and the number of stairs walked per day, and the Nike+ Fuelband[3] which aggregates such individual measures into a proprietary notion of fuel. Given that participants in our two studies did assert interest in measuring their productivity, what might similar individual or aggregate measures be that are useful for tracking and reporting to software developers?

Although developers in our survey suggested many possible individual measures (Figure 2), further investigation of the top ranked item in our observational study indicated the difficulty of using one simple metric, in this case, number of work items closed. Taken together, the results of the two studies suggest that there might not be a single and/or simple measure for a developer's productivity. For example, while summarizing the time developers spent on certain activities during the day might provide some insight, as for example done by Codealike[4], to be more meaningful it has to be combined with measures that also provide a certain, possibly personal, assessment of the productivity of the time spent, such as the progress made towards certain tasks, the value of the task for the customers, or how focused the developer was during the time. Based on a combination of such measures, some of which might be qualitative rankings by a developer, one might be able to abstract these to an overall productivity level for a day, not unlike the Nike fuel idea. Seeing fluctuations in an abstracted measure may provide sufficient support for a developer to retrospect on how work was performed, particularly if detailed information could be provided, such as the visualization of task and program switches in Figure 3, to support the retrospection.

We believe the benefit of enabling and driving retrospection may be the key rather than trying to define measures that can be compared across individuals or across organizations. By driving productivity for the individual in a personalized way, we can also help assuage privacy concerns mentioned by the participants in our studies with who might be able to see productivity information and to whom it might be compared, such as co-workers. We plan to investigate aggregate approaches to visualizing and reporting productivity that are driven by the individual and assess how the provision of such information enables retrospection and hopefully drives up productivity.

Reducing Context Switches.

Participants in our study mentioned that being focused and "in the flow" without context switches increases their productivity. We can see benefits in visualizing context switches to developers and using rates or measures based on context switches as part of an aggregate measure of productivity in a given time period. However, this raises the question of how one can automatically identify context switches in a developer's work given that the switch is based not only on the program used, not only on the task defined, but on the semantics of the work the developer is performing. For instance, a developer might be switching to respond to a personal email after writing an email to a colleague about a problem in the code, thus staying in the same program but still switching context. Participants in our observational study clearly stated that quick context switches to perform such a task, as reading email while waiting for a build, did not impact productivity but switches that require a change in thinking are most costly and do impact productivity. Investigation is needed to determine whether it is possible to correlate program switches to help enable the determination of the cost of a context switch, for instance, a switch from a program that enables builds to be launched to email may be considered a low cost context switch. Investigation is also needed to determine if analysis of the information in programs used adjacently can help in assessing the cost of a context switch. Perhaps the combination of the correlation of program usage with the correlation of information in adjacently used programs can help assess whether it is more likely that the switch is between semantically different tasks rather than just a program switch and thus more likely to have a high cost. This information could be helpful in the development of aggregate and individual productivity measures and could be helpful to support developers in reflecting upon their work habits.

Even without knowledge of the exact cost of a specific context switch and without being able to retrospectively analyze one's productivity in terms of context switch behavior, there are multiple strategies that can help to avoid or prevent context switches. The ones mentioned by our study participants ranged from closing the email client and shutting off notifications, to listening to music, closing the office door, scheduling a meeting with oneself, coming in early to work, or working from home. One team even used an explicit indicator for avoiding work disruptions by others at certain times:

[..] all devs in our team also have a physical light on top of our monitors that reflects our lync status, so people walking by can see not to disturb. This has been really useful in reducing random interruptions. (MS5)

There is a trade-off though between an individual developer's productivity gain by avoiding interruptions from coworkers and a team's productivity loss. With a better assessment of the impact a context switch would have on a developer's productivity, one could devise approaches to optimize the productivity gain.

[2] `fitbit.com`, verified 03/15/14
[3] `www.nike.com`, verified 03/15/14
[4] `codealike.com`, verified 03/15/14

Setting Goals.

The setting of goals is another technique that can be used to motivate and enforce a behavior change. For example, goal-setting has been shown to be successful in motivating people to be more active, in particular in combination with self-monitoring (e.g. [10, 7]). When it comes to software development, many of our participants stated they set goals for themselves on a daily or weekly basis. Although the developers did not necessarily believe that goal-setting helped them increase their productivity, they stated that the goals provide an overview of their tasks, allow them to prioritize work and to better react in cases of unplanned tasks, which were mentioned to be a major detriment to being and feeling productive. Goals are often written down in a simple todo list and allow developers to measure their progress and productivity, creating a certain happiness and satisfaction when the goals are met. However, as with a more general retrospection of a developer's productivity, monitoring goals either requires manual work by the developer or measures that can be tracked automatically, and describe the progress towards fulfilling a goal.

As with monitoring, several participants fear the negative effect goal monitoring can have and stated that the additional time needed for monitoring could better be used for actual work. From talking to developers, it seems however that the overhead of setting goals, is negligible and well worth it since it helps to better structure the next workday and thus make them more focused. This approach is a best practice that could be shared amongst developers.

6. CONCLUSION

To investigate how developers perceive and assess their own productivity, we conducted two studies, a survey with 379 professional software developers and an in-person observation study with 11 professional developers in three different companies. The survey results show that while most participants considered coding highly productive, there are several activities, such as meetings and handling emails, that are more difficult for developers to assess with respect to productivity. The results also show that while progress made on work items is considered as one of the better measures to assess productivity, there is no simple and single best measure to use. Developers also like to organize their work to get in "the flow" so as to have few interruptions and context switches. Interestingly, the observation data we collected paints a different picture, showing that developers experience a high number of switches in their work, switching tasks every 6.2 minutes and activities every 1.6 minutes. The reason why developers still feel productive is the varying cost associated with these varying forms of context switches.

Based on this data, we propose a number of ways to improve a developer's productivity through tool support and the sharing of best practices. For instance, tool support to recognize context switches and visualize the switches to a developer may help a developer in retrospecting on their own productivity. There are also interesting avenues to explore in terms of helping developers define aggregate personal productivity measurements for tracking and retrospection. With more support for retrospection on productivity, developers may be able to better share best practices, such as goal setting best practices discussed by participants in our observational study.

7. ACKNOWLEDGMENTS

The authors would like to thank the participants in the survey and observational study. The authors would also like to thank Chris Bird for his help and the anonymous reviewers for their helpful feedback. This work was funded in part by NSERC.

8. REFERENCES

[1] www.ifi.uzh.ch/seal/people/meyer/developers-productivity.

[2] M. Andreessen. Why software is eating the world. The Wall Street Journal, August 20, 2011.

[3] K. Beck and C. Andres. *Extreme programming explained: embrace change*. Addison-Wesley, 2004.

[4] A. Begel and B. Simon. Novice software developers, all over again. In *Proceedings of the Fourth International Workshop on Computing Education Research*, ICER '08, pages 3–14. ACM, 2008.

[5] J. Blackburn, G. Scudder, and L. Van Wassenhove. Improving speed and productivity of software development: a global survey of software developers. *IEEE Transactions on Software Engineering*, 22(12):875–885, 1996.

[6] B. W. Boehm. Improving software productivity. volume 20, pages 43–57. IEEE, 1987.

[7] D. M. Bravata, C. Smith-Spangler, V. Sundaram, A. L. Gienger, N. Lin, R. Lewis, C. D. Stave, I. Olkin, and J. R. Sirard. Using pedometers to increase physical activity and improve health: A systematic review. *Jama*, 298(19):2296–2304, 2007.

[8] D. N. Card. The Challenge of Productivity Measurements. In *Pacific Northwest Software Quality Conference*, pages 1–10, 2006.

[9] 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 *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '08, pages 2–11. ACM, 2008.

[10] S. Consolvo, P. Klasnja, D. W. McDonald, and J. A. Landay. Goal-setting considerations for persuasive technologies that encourage physical activity. In *Proceedings of the 4th international Conference on Persuasive Technology*, pages 8:1–8:8. ACM, 2009.

[11] S. Consolvo, D. W. McDonald, T. Toscos, M. Y. Chen, J. Froehlich, B. Harrison, P. Klasnja, A. LaMarca, L. LeGrand, R. Libby, et al. Activity sensing in the wild: a field trial of ubifit garden. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1797–1806. ACM, 2008.

[12] L. Dabbish, G. Mark, and V. M. González. Why do i keep interrupting myself?: Environment, habit and self-interruption. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 3127–3130. ACM, 2011.

[13] C. R. B. de Souza and D. F. Redmiles. An empirical study of software developers' management of dependencies and changes. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 241–250. ACM, 2008.

[14] T. DeMarco and T. Lister. Programmer performance and the effects of the workplace. In *Proceedings of the 8th International Conference on Software Engineering*, ICSE '85, pages 268–272. IEEE, 1985.

[15] P. Devanbu, S. Karstu, W. Melo, and W. Thomas. Analytical and empirical evaluation of software reuse metrics. In *Proceedings of the 18th International Conference on Software Engineering*, ICSE '96, pages 189–199. IEEE, 1996.

[16] E. W. Dijkstra. The humble programmer. *Communications of the ACM*, 15(10):859–866, 1972.

[17] B. J. Fogg. Persuasive technology: Using computers to change what we think and do. *Ubiquity*, 2002:5, 2002.

[18] T. Fritz, E. M. Huan, 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 *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014. to appear.

[19] W. Gibbs. Software's chronic crisis. *Scientific American*, 271(3):86–94, 1994.

[20] V. M. González and G. Mark. "constant, constant, multi-tasking craziness": Managing multiple working spheres. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 113–120. ACM, 2004.

[21] H. Hulkko and P. Abrahamsson. A multiple case study on the impact of pair programming on product quality. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 495–504. ACM, 2005.

[22] W. S. Humphrey. *Introduction to the Personal Software Process*. Addison-Wesley Professional, first edition, 1996.

[23] W. S. Humphrey. Using a defined and measured personal software process. *IEEE*, 13(3):77–88, 1996.

[24] P. M. Johnson, H. Kou, J. Agustin, C. Chan, C. Moore, J. Miglani, S. Zhen, and W. E. J. Doane. Beyond the personal software process: Metrics collection and analysis for the differently disciplined. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 641–646. IEEE, 2003.

[25] C. Jones. Software metrics: good, bad and missing. *Computer*, 27(9):98–100, 1994.

[26] D. Kamma and P. Jalote. Effect of task processes on programmer productivity in model-based testing. In *Proceedings of the 6th India Software Engineering Conference*, ISEC '13, pages 23–28. ACM, 2013.

[27] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '06/FSE-14, pages 1–11. ACM, 2006.

[28] R. B. Kieburtz, L. McKinney, J. M. Bell, J. Hook, A. Kotov, J. Lewis, D. P. Oliva, T. Sheard, I. Smith, and L. Walton. A software engineering experiment in software component generation. In *Proceedings of the 18th International Conference on Software Engineering*, ICSE '96, pages 542–552. IEEE, 1996.

[29] A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 344–353. IEEE, 2007.

[30] J. A. Lane and D. Zubrow. Integrating measurement with improvement: An action-oriented approach: Experience report. In *Proceedings of the 19th International Conference on Software Engineering*, ICSE '97, pages 380–389. ACM, 1997.

[31] H. Mintzberg. *The nature of managerial work*. Theory of management policy series. Prentice-Hall, 1980.

[32] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002.

[33] S. Munson. Mindfulness, reflection, and persuasion in personal informatics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2012.

[34] P. Naur and B. Randell. Software engineering: Report of a conference sponsored by the nato science committee. *Scientific Affairs Division, NATO*, 1969.

[35] V. Nguyen, L. Huang, and B. Boehm. An analysis of trends in productivity and cost drivers over years. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, Promise '11, pages 3:1–3:10. ACM, 2011.

[36] D. E. Perry, N. A. Staudenmayer, and L. G. Votta. People, organizations, and process improvement. *Software, IEEE*, 11(4):36–45, 1994.

[37] O. U. Press. Oxford dictionary. www.oxforddictionaries.com/us/definition/american_english/productivity.

[38] T. Roehm, R. Tiarks, R. Koschke, and W. Maalej. How do professional developers comprehend software? In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 255–265. IEEE, 2012.

[39] J. Sillito, G. C. Murphy, and K. De Volder. Questions programmers ask during software evolution tasks. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '06/FSE-14, pages 23–34. ACM, 2006.

[40] S. Wagner, M. Ruhe, and A. Siemens. A systematic review of productivity factors in software development, 2008.

[41] M. Zhou and A. Mockus. Developer fluency: Achieving true mastery in software projects. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '10, pages 137–146. ACM, 2010.