Detecting Developers' Task Switches and Types

André N. Meyer, Chris Satterfield, Manuela Züger, Katja Kevic, Gail C. Murphy, Thomas Zimmermann, and Thomas Fritz

Abstract—Developers work on a broad variety of tasks during their workdays and constantly switch between them. While these task switches can be beneficial, they can also incur a high cognitive burden on developers, since they have to continuously remember and rebuild the task context–the artifacts and applications relevant to the task. Researchers have therefore proposed to capture task context more explicitly and use it to provide better task support, such as task switch reduction or task resumption support. Yet, these approaches generally require the developer to *manually* identify task switches. Automatic approaches for predicting task switches have so far been limited in their accuracy, scope, evaluation, and the time discrepancy between predicted and actual task switches. In our work, we examine the use of *automatically* collected computer interaction data for detecting developers' task switches as well as task types. In two field studies–a 4h observational study and a multi-day study with experience sampling–we collected data from a total of 25 professional developers. Our study results show that we are able to use temporal and semantic features from developers' computer interaction data to detect task switches and types in the field with high accuracy of 84 percent and 61 percent respectively, and within a short time window of less than 1.6 minutes on average from the actual task switch. We discuss our findings and their practical value for a wide range of applications in real work settings.

Index Terms—Task detection, task switching, multi-tasking, work fragmentation, activity recognition, machine learning

1 Introduction

To successfully perform their work, software developers are required to constantly switch between a broad variety of tasks, such as implementing a new feature, answering an email or attending a meeting, with each task requiring its own set of artifacts and applications [1], [2], [3]. These constant task switches result in a high fragmentation of work, requiring developers to continuously interrupt and later resume their tasks and to relocate the artifacts and applications that are relevant to fulfill the task at hand. Subsequently, developers face a higher cognitive burden, lower performance, and a higher error rate [4], [5].

To support developers in their fragmented task work, researchers have proposed approaches that explicitly capture task context–artifacts and applications relevant to the task–and that use this information to then support users by preventing interruptions, easing task resumption, or by recommending relevant artifacts and applications [6], [7], [8], [9], [10], [11]. While studies have shown that the explicitly captured task context can lower the cognitive burden on developers and increase productivity [6], [12], all of these approaches require some form of *manual* interaction of the developer to identify task boundaries, something that developers often forget to do in practice after using such an approach for a few days [6].

To address this issue, few researchers have proposed approaches to *automatically* detect switches between tasks, varying mainly in the features used (e.g.,user input or application based), and the method applied (e.g.,supervised versus unsupervised machine learning) [13], [14], [15]. Yet, the evaluations performed to study these approaches are often fairly limited in terms of the tasks and number of participants, and the results show that it is very challenging to achieve high prediction accuracy of task switches without too many false positives [15], [16], [17], or that one has to accept a high deviation in time of 3 to 5 minutes between predicted and actual task switches [13], [14], [18]. Since these approaches focus on detecting task switches within the IDE only, they are not capturing non-development work, which can account for 39 percent up to 91 percent of the time developers spend at work [1], [3], [19], [20], [21].

In our research, we extend this work and investigate *(RQ1)* whether we can automatically detect task switches of professional software developers in the field, based on temporal and semantic features as extracted from their computer interaction inside and outside the IDE. We were also interested in classifying the type of task a developer is working on, since the better we understand the context of a task, the better we can support developers. To the best of our knowledge, there has been only one approach so far that looked at the *automatic* classification of developers' activities on a task level [22]. Yet, their examination was limited to specific development activities only, and did not consider the whole range of non-development tasks that developers are working on, such as administrative or planning tasks. In our work, we investigate the task types that software developers are working on more holistically,

- *Andre N. Meyer, Manuela Züger, and Thomas Fritz are with the Department of Informatics, University of Zurich, 8006 Zürich, Switzerland. E-mail: {ameyer, zueger, fritz}@ifi.uzh.ch.*
- *Chris Satterfield and Gail C. Murphy are with the University of British Columbia, Vancouver, BC V6T 1Z4, Canada. E-mail: {cds00, murphy}@cs.ubc.ca.*
- *Katja Kevic and Thomas Zimmermann are with Microsoft Research, Redmond, Washington USA. E-mail: {kakevic, tzimmer}@microsoft.com.*

Manuscript received 13 June 2019; revised 19 Mar. 2020; accepted 19 Mar. 2020. Date of publication 16 Apr. 2020; date of current version 10 Jan. 2022. (Corresponding author: André N. Meyer.) Recommended for acceptance by D. Poshyvanyk. Digital Object Identifier no. 10.1109/TSE.2020.2984086

and explore *(RQ2)* how accurately we can predict them in the field.

To address our research questions, we performed two field studies: one with 12 professional developers in which we observed their work over a 4-hour period and logged the task switches and types without interrupting their work; and one with 13 professional developers in which we regularly prompted participants to self-report their task switches and types over a period of about 4 workdays and conducted a post-study questionnaire. By varying the study methods, we wanted to achieve a higher generalizability of our results and ensure that we take into account the effects of self-reporting while also capturing the breadth of developers' tasks over multiple days. For both field studies, we also collected the participants' computer interaction using a monitoring tool that we installed on their machine and that was running in the background. From the computer interaction data, we extracted a total of 109 temporal and semantic features. Our analysis of the data shows we can use the automatically logged computer interaction data to train machine learning classifiers and predict task switches with a high accuracy of 87 percent, and within a short time window of less than 1.6 minutes of the actual task switch. Our analysis further shows that we are able to predict task types with an accuracy of 61 percent, yet that this accuracy varies a lot by task type. The features based on mouse and keyboard interaction generally hold the highest predictive power, while the lexical features we extracted from the application names and window titles have the least predict power in our approach.

Overall, our work extends previous work with an approach that uses a broader range of features, also works outside the IDE to capture developers' work more holistically, is evaluated in a field-study with 25 professional developers, and achieves higher accuracy and less delay than previous work. Our results provide evidence for the potential to *automatically* detect software developers' task switches and types in the field. This opens up opportunities for providing developers with task support tools that lower the burden of fragmented work and constant task switching, by reducing task switching and facilitating task resumption, and by greatly complementing existing task support by freeing the developer from the laborious manual task boundary identification.

The primary contributions of this paper are:

- An approach to automatically detect task switches and types based on developers' computer interaction that is not limited to the IDE.
- Two field studies with 25 professional developers demonstrating our approach's potential to detect task switches and types with high accuracy and within a small time window in the field.
- An evaluation of the predictive power of various computer interaction features, including semantic and temporal ones, and a comparison of individually trained models versus a general model.

2 Related Work

Work related to our research can broadly be grouped into research that examined the detection of task switches and task types, and into approaches to support task focused work. Based on previous work [2], [6], [23], [24], [25], [26], we defined a task as a *well-defined work assignment with a specific goal that people divide their work into*, such as fixing a bug, or preparing for a team-meeting. A *task switch* occurs, when a person switches between two different tasks.

2.1 Task Switch Detection

Several researchers have explored the detection of task switches mostly for general knowledge workers. These approaches mainly differ in the features they used to identify the task boundaries or switches, ranging from semantic features to temporal features, the method they use, unsupervised versus supervised, and the way they evaluated their approach. One of the most prominent approaches is by Shen *et al.* [13], [14], [16], [18] that is mainly based on semantic features and supervised learning. They reused an approach, TaskTracer [7], that allows users to manually indicate the tasks they are working on, and additionally tracks their application interactions in the background, including window titles. Based on the assumption that windows of the same task share common words in their titles, they create vectors from window titles and identify task switches based on a textual similarity measure using the users' previously declared tasks and supervised learning. After the first version [18], they further improved their approach to reduce the number of false positives and to be able to predict task switches online [13], [14]. Their evaluation is based on a small set of two users and counts a task switch as accurate if it falls within a 4 to 5 minute time window of a real switch, which is a very coarse measure, given the frequent task switching in today's environment that happen every few minutes [2], [23]. Based on the assumption that switches between windows of the same task occur more frequently in temporal proximity than to windows of a different task, Oliver *et al.* [27] examined a temporal feature of window switches within a 5 minute time window in addition to semantic features and using an unsupervised approach. An evaluation based on 4 hours of a single participant, showed a precision of 0.49 and recall of 0.72. Researchers have also used other temporal features, in particular, the frequency of window events, to determine task switches. Under the assumption that users navigate between windows more frequently when they switch tasks, as opposed to during a task, Nair *et al.* [17] developed a system that calculates window event frequency based on fixed 5 minute time windows. An evaluation with 6 participants resulted in an accuracy of 50 percent. Mirza *et al.* [15] relaxed the constraint of a fixed time window, used adjusted frequency averages and studied the various approaches with 10 graduate students. They found that their approach improved the accuracy and achieved an overall accuracy of 58 percent. Overall, previous research has shown that detecting task switches is difficult, even for very short periods of time and in controlled environments. In our work, we focus on software development work and extend these approaches by including and examining both, semantic and temporal features of window events as well as user input features, and by conducting two studies with professional software developers.

Only little research has been performed on task switch detection in the software development domain and all of this

research has focused solely on software development tasks within the IDE. As one of the first, Robillard and Murphy [28] proposed to use program navigation logs to infer development tasks and they built a prototype, however, without evaluating it. In 2008, Coman and Sillitti [29] focused on splitting development sessions into task-related subsections based on temporal features of developers' access to source code methods and evaluated their approach with 3 participants over 70 minutes each, finding that they can get close to detecting the number of task switches, yet the point in time when the task happens is a lot more difficult. Zou and Godfrey [30] replicated Coman and Sillitti's study in an industrial setting with six professional developers and found that the algorithm detects many more task switches than the ones self-reported by the participants with an error of more than 70 percent. Finally, on a more fine-grained level, Kevic and Fritz [31] examined the detection of activity switches and types within a change task using semantic, temporal and structural features. In two studies with 21 participants, they found that activity switches as well as the six self-identified activity types can be predicted with more than 75 percent accuracy. Different to these approaches, we focus on all tasks a developer works on during a day, not just the change tasks within the IDE.

2.2 Task Type Detection

Researchers also examined detecting the type of task or activity a person is working on. Most similar to our approach for task type detection is Koldijk *et al.* [32]. They investigated the use of features over a fixed 5 minute time window, using mouse and keyboard input, application (switches) and the time of day. They tried to predict one of 12 task types that they identified in a survey, e.g., read email, write email, plan, program, search information, and create visualization. The results of a field study with 11 researchers and an average of 10 hours of data per participant shows that the prediction is very individual and that a general classifier does not work well. Mirza *et al.* [33] focused on classifying users' desktop interactions into six higher level activity types (not task types): writing, reading, communicating, web browsing, system browsing and miscellaneous. They used temporal, interaction-based (application window events), and semantic features calculated over a 5 minute time window. In a 6 hour field study with five participants and a controlled lab study, they found that they can predict the activity category for each of these 5 minute windows with high accuracy (81 percent) and that interaction-based features work best. Researchers have also explored biometric features, such as Hassib *et al.* [34] who used Electroencephalography to classify the task type according to cognitive load, but without looking at specific task types. In our work, we extend these approaches by not fixating on fixed time windows of 5 minutes but by first detecting the actual switches and then evaluating them with professional developers in the field.

We have been able to find only very little research on predicting task types for software developers. To the best of our knowledge, the only approach that is similar to our work is by Bao *et al.* [22]. In their work, they automatically track low-level computer interaction data (user input and application usage) and use a Conditional Random Field (CRF) based approach to segment the data and infer one of six development activities–coding, debugging, testing, navigation, search, or documentation–similar to our task types. An analysis of data collected from ten developers over a week shows that CRF is able to classify the activity with 73 percent accuracy. We extend the approach by focusing on all activities developers perform during their workday, not just development, and by examining when developers switch between different tasks.

2.3 Task Support

While there is a vast number of approaches to support specific development activities, such as code search, code review or debugging, only little research has looked into supporting developers with understanding and managing their tasks and the frequent switches between them. Several researchers have therefore proposed to explicitly model development tasks and to capture task contexts–artifacts and applications relevant to a task–to support developers in their task work, in particular by recommending relevant artifacts [6], [8], identifying related tasks [35], easing the resumption of interrupted tasks and switching between them [6], [7], [9], [10], [11], [12], [36], [37], [38], or scoping queries and recommending workflow improvements [5]. Early approaches to support task switching, such as virtual workspaces [10] or the GroupBar [9], provide interfaces that allow the user to manually group artifacts and applications with respect to tasks. The approaches Mylyn by Kersten and Murphy [6], and TaskTracer by Dragunov *et al.* [7], both explicitly capture task context by automatically recording user interactions within the IDE or the desktop environment respectively, given the user manually indicates the start and end of a task. While several of these approaches have great potential to support developers in their task work, they require some form of *manual* interaction to identify the task boundaries, something that developers often forget to do after using such a tool for a while [6]. Researchers have therefore examined how to best aid developers in identifying task boundaries retrospectively [39], looked into more lightweight approaches for supporting task resumption through cues and without specific task context [40], or explored the *automatic* mining of task contexts to support window switching [38] and grouping files [37]. Overall, an automatic and real-time task switch detection has thereby the potential to complement and significantly improve the value of most of these existing approaches for developers.

3 Study Design

To investigate the use of computer interaction data for predicting task switches and types, we conducted two field studies, a 4-hour observational study and a multi-day study with experience sampling, with a total of 31 professional software developers initially. The observations and self-reports served as the ground truth of participants' task switches and types, while we additionally gathered computer interaction data to extract features for our predictions. In both studies, we used the same definitions of tasks, task switches and types which we also shared with the participants. A brief overview of our study design is presented in Fig. 1.

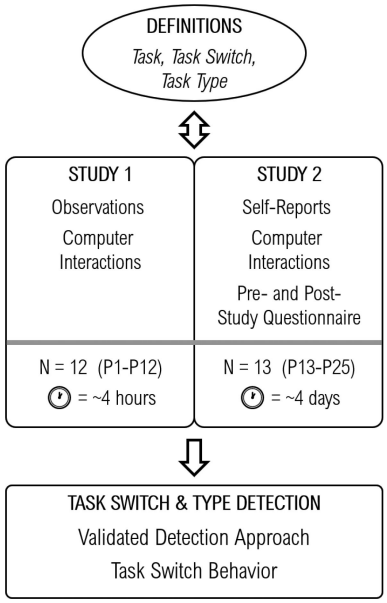

Fig. 1. Overview of the study design and outcomes.

3.1 Study 1 – Observations

In our first study, we observed the work of 12 participants over a period of 4 hours to gather a richer understanding of developers' task switches and types they work on.

Procedure. For the observations, the observer, either the first or second author, followed a detailed protocol that we developed before the study. The very first observation session was performed by both observers at the same time. A cross-check of the two observation logs showed an inter-rater agreement of 97 percent, suggesting a high overlap of observing the same tasks and task switches.

Before each observation session, the observer explained the study purpose and process to the participants and asked them to sign a consent form, to install a *monitoring tool* that tracks participants' computer interaction, and to describe the tasks they were planning to work on during the observation. The observer also introduced herself to nearby colleagues and asked them to ignore her as much as possible, and collaborate with the observed participant as they would normally do. After that, the observer placed herself behind the participant to prevent distractions, while still being able to see the screen contents on the participant's computer. Finally, the observer started the actual observation session and asked the participant to continue their work as usual.

We observed participants for a total of four hours each on a single workday: two hours before and two after lunch. For the observations, we followed Mintzberg's protocol of a structured observation session [41]. The observer wrote in an observation log[1] each time the participant switched from one task to another. Each entry in the observation log consists of a timestamp, a description of the reason for the task switch and a description of the task itself. We inferred tasks and their details from the active programs and their contents on the screen, as well as discussions participants had with co-workers. After each session, the observer validated the observed tasks and task switches with the participant, by going through the list of observed tasks and accompanying notes and modifying mistakes made during the observation.

Participants. We recruited 14 participants through professional and personal contacts from two large-sized and one medium-sized software companies. We excluded two participants for which we were not able to observe a sufficient amount of task switches (less than 10). Of the remaining 12 participants, 1 was female and 11 were male. Throughout the paper, we refer to these participants as P1 to P12. Our participants had an average of 10.8 ($\pm$7.4, ranging from 1 to 20) years of professional software development experience and were working in different roles: 8 participants identified themselves as individual contributors and 4 as developers in a leading position. All participants resided either in Canada or the United States.

Monitoring Tool. To collect computer interaction data from developers, we developed and used our own monitoring tool, PersonalAnalytics[2], for the Windows operating system. The tool tracks participants' mouse and keyboard interaction, as well as their application usage. For the mouse, the tool tracks the clicks (coordinates and button), the movement (coordinates and moved distance in pixels), and the scrolling (coordinates and scrolled distance in pixels) along with the corresponding time-stamp. For the keyboard, the tool records the type of each keystroke (regular, navigating, or backspace/delete key) along with the corresponding time-stamp. For privacy reasons, we did not record specific keystrokes. Our tool further records the currently active application, along with the process name, window title, and time-stamp whenever the window title changed or the user switched to another application.

Task Type Inference. We inferred task type categories by performing a Thematic Analysis [43] on the basis of related work and our observation logs. The analysis process included first familiarizing ourselves with the observed task switches, open coding the observed and participant-validated tasks and accompanying notes, identifying themes, and categorizing the resulting themes into higher level task types. This process resulted in nine task type categories: *Development*, *Personal*, *Awareness & team*, *Administrative*, *Planned meeting*, *Unplanned meeting*, *Planning*, *Other* and *Study*. The task types are described in more detail in Table 4 and discussed in Section 6.1. In contrast to a task (and the task type), an activity describes an event or happening that does not necessarily need to have a particular purpose (or task). For example, the activity *Web Browsing*, could be grouped into several task types, such as *Development* when the developer is reading API documentation online and *Planned meeting* when the developer is using an online-conferencing tool.

3.2 Study 2 – Self-Reports

To capture a longer time period and more breadth in developers' work, we conducted a second field study with 13 participants over a period of 4 workdays each. For this study, we used experience sampling, in particular we regularly prompted participants to self-report task switches and types. By using experience sampling, we also wanted to

1. We used our own observation logging tool: https://github.com/casaout/ObservationStudyTool

2. https://github.com/sealuzh/PersonalAnalytics. Details can be found in [42].

mitigate the risk of a bias in participants' behavior due to an observer sitting behind them, which, for example, could lead to participants being less likely to browse work unrelated websites.

Procedure. Before the study, we emailed participants a document explaining the study goal and high-level procedure, asked them to sign a consent form and to answer a pre-study questionnaire with questions on demographics, their definition of a task, reasons for switching between tasks, and on the task types they are usually working on. Afterwards, participants received the study instructions, detailing the study goals, definitions of task switches and types that we used for the study, and instructions on how to install and run the monitoring tool. They were asked to install the same *monitoring tool* that we described above on their main computer. In case participants worked on multiple computers (e.g.,a desktop and a laptop), we asked them to install the monitoring tool on both devices. Participants were further asked to read our definitions of a task, task switch and task type, as well as instructions on how to use the *self-reporting component* that we added to our monitoring tool. Finally, participants were asked to pursue their work as usual for the next couple of workdays while also self-reporting their task switches and types when the pop-ups/prompts appeared.

For this study, our tool prompted participants once per hour to self-report their task switches and types for the previous hour. The self-reporting step is explained in more detail below. We intentionally decided to use an interval of one hour rather than a full day, to balance the intrusiveness of the prompts with the ability to accurately remember tasks and task switches over the previous time interval [44]. To further ensure high quality in the collected self-report data we further allowed participants to withdraw from the study at any point in time, and to pick the time for their participation themselves. In addition, and to avoid boredom or fatigue, we asked participants to respond to a total of 12 to 15 prompts, assuming an average of four self-reports per day and a total of three to four workdays for participation. This number was a result of several test-runs over multiple weeks and from qualitative feedback gathered with a pilot participant, a professional developer. Furthermore, we provided support to postpone self-report prompts for 5 minutes, 15 minutes, or 6 hours, and built and refined the self-reporting component to require as little effort as possible to answer, e.g., by letting participants answer the required fields by simply clicking on elements instead of asking them for textual input. Finally, each pop-up also asked participants to report their confidence with their self-reports.

Throughout the study, participants could check the number of completed pop-ups. Once they completed 12 pop-ups, participants could notify us and upload the collected data and self-reports to our server. The upload wizard once again described the data collected and allowed participants to obfuscate the data before sharing it with us. At the end of the study, participants were asked to answer a post-study questionnaire with questions on the experienced difficulties when self-reporting task switches and task types, on further task types they were working on, and on how they could imagine using information on task switches and types. After completing the survey, participants were given two 10 US$ meal cards to compensate for their efforts.

Participants. We recruited 17 participants through professional and personal contacts from one large-sized software company. We discarded data from three participants that self-reported less than 10 task switches in the days of their participation. We further discarded the data of one participant whose definition of a task switch was very different to ours and the rest of the participants (i.e., he considered every application switch a task switch). Of the remaining 13 participants that we used for the analysis, 2 were female and 11 were male. Our participants had an average of 12.1 ($\pm$8.2, ranging from 1 to 30) years of professional software development experience and were working in different roles: 10 identified themselves as individual contributors and 3 as developers in a leading position (i.e., Lead or Manager). All participants resided in the United States. In the paper, we refer to these participants as P13 to P25.

Self-Reporting Component. The self-reporting component is part of our monitoring tool and includes a pop-up with three pages. The *first* page asked participants to self-report the task switches they experienced in the past hour. It visualized participants' application usage on a timeline using different colors for each application and allowed them to self-report their task switches by clicking on the lines denoting applications switches. We restricted the task switch self-reports to a granularity of application switches with a minimum length of 10 seconds for a variety of reasons: First, we assumed that most of participants' task switches coincide with application switches (e.g.,switching from the email client to the IDE, or from the browser to an IM client) and fewer happen during a session uniquely spent within the same application (e.g.,switching tasks directly in the IDE or in the browser). And, we wanted to avoid cluttering the user interface of our self-reporting component and simplify the reporting for participants. Similar to [14], the timeline visualization provided additional details when the participant hovers over an application, such as the application name, time when it was used, window title(s) and user input produced in that application. As soon as participants completed self-reporting their task switches for the whole previous hour, they could proceed to the *second* page and self-report their task types (see Fig. 2). On the second page, we visualized the same timeline as before, but added another row that prompted participants to select task types from a drop-down menu. After selecting the task types for all task segments, participants could proceed to the last page. The *third* page asked participants to self-report their confidence with their self-reports of task switches and task types on a 5-point Likert-scale (5: very confident, 1: not at all confident) and optionally add a comment. Capturing participants' confidence served as an indicator of the quality and accuracy of their self-reports. The user interface we used to collect the ground truth for task switches and types resembles the one by Mirza *et al.* [15], [25], [33].

The supplementary material [45] includes the study instructions we shared with participants, the pre- and post-study questionnaires they answered and additional screenshots detailing the self-reporting component.

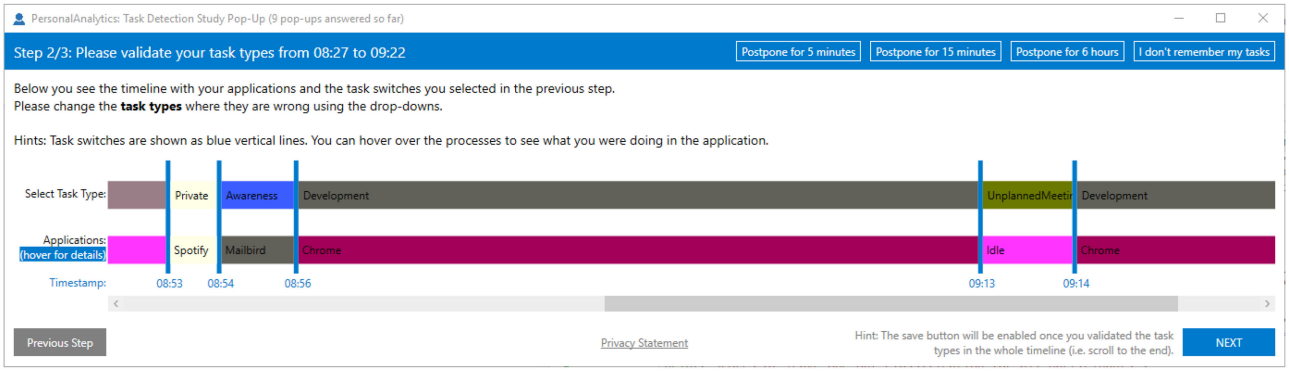

Fig. 2. Screenshot of the second page of the experience sampling pop-up that asked participants to self-report their task types.

4 Data and Analysis

For this study, we collected two rich data sets, including observed or self-reported ground truth data, and automatically tracked computer interaction data. Prior to the main analysis of the data, we performed multiple pre-processing steps, including data segmentation and feature extraction, which are summarized in the remainder of this section.

4.1 Collected Data

For our *study 1*, we collected observation logs for a total of 51.7 hours of work and an average of 4.3 (±1.3) hours per participant. For our *study 2*, we collected self-reports for a total of 58 workdays and an average of 4.5 (±1.7) days per person. On average, participants reported a high confidence with their self-reports (>3) in 20.6 (±9.0), and a medium or low confidence (≤ 3) in 22.2 (±16.7) of the pop-ups they answered. 77.0 percent was the highest ratio of medium or low confidence self-reports that one participant had, and 16.7 percent was the lowest. We decided to only use the data of the 268 self-reports with a high confidence (>3), thus including a total of 268 hours of work and discarding the rest (289 self-reports). This allowed us to ensure we were training our models with data that is of high quality and accuracy. Future work could also account for over- or under-confidence in participants' self-reports.

Table 1 reports statistics on the self-reports. Since overall, only 11 percent of the pop-ups were postponed by participants, one reason for the relatively high number of self-reports with medium or low confidence could be that the pop-ups appeared at inopportune moments and participants did not remember they could postpone it. Instead, participants might have just clicked through the pop-up and reported a low confidence to not distort the data. We discuss possible threats in Section 7 and improvements in Section 8.

TABLE 1
Self-Reports for Study 2

	All	per Part.
Days participated	58	4.5 (±1.7)
Pop-ups displayed to participants	557	42.8 (±21.6)
Pop-ups answered by participants	268	20.6 (±9.0)
- Pop-ups answered within 5 minutes	158	12.2 (±6.3)
- Pop-ups answered after 5 minutes	110	8.5 (±5.4)
Pop-ups postponed by participants	62	4.8 (±3.5)
Pop-ups discarded by researchers	289	22.2 (±16.7)

4.2 Time Window Segmentation

To calculate and extract *task switch detection* features, we defined the time windows to be between two application switches, which we call *application segments*. Thus, the task switch detection model that we were going to build, could recognize task switches whenever a developer switches between applications, but would miss task switches within an application, such as a switch from a work item to the next one inside the IDE. We consider application segments to be an appropriate time window with minimal prediction delay, since developers spend on average only 1 to 2 minutes in an application before switching to another [2], [3], [23], and to ensure the accuracy of participants' self-reports (in study 2) was high. Threats to this classification are discussed in Section 8. In contrast, previous approaches predominantly used longer and fixed window lengths of 5 or 10 minutes [13], [14], [17], [27]. These shorter and more flexible time windows at borders of application switches allow to more accurately capture developers' behaviors, and to more precisely locate the point in time of the task switch. For the *task type detection* features, we used the time windows between two task switches, as identified by our observations (study 1) or participants' self-reports (study 2), which we call *task segments* for the feature extraction.

4.3 Task Switch Features Extracted

A next step towards building a classifier for task switch detection is to extract meaningful features from the raw computer interaction data collected by the monitoring tool. The features that we developed are either based on heuristics that participants stated as indicative of their task switches in the post-study questionnaire (study 2), based on features that have been linked to developers' task switching behavior in prior work, as well as based on our own heuristics. The features we used are presented in Table 2 and are discussed in more detail in the remainder of this section.

Task switch detection is a special case of change-point detection [49], [50], which is the process of trying to detect abrupt changes in time-series data. This is why many of our features compare the similarity between characteristics of the previous application segments with the current one, for example the difference in the number of keystrokes. To determine how many steps back one needs to compare the

TABLE 2
Features Analyzed in Our Study and Their Importance for Predicting Task Switches and Task Types

Features	Import. Switch	Import. Type *All*	Import. Type *UI*
User Input Features	**45.8%**	**52.0%**	**81.0%**
Keystroke differences (4): difference in the number of navigate/backspace/normal/total keystrokes pressed per second between the previous and current application/task segment [22], [32], [33], [46]	16.4%	19.1%	29.9%
Mouse click differences (4): difference in the number of left/right/other/total mouse clicks per second between the previous and current application/task segment [22], [32], [46]	17.9%	13.9%	29.4%
Mouse moved distance (1): total moved distance (in pixels) of the mouse per second [46]	6.8%	8.7%	13.5%
Mouse scrolled distance (1): total scrolled distance (in pixels) of the mouse per second [22], [32]	4.7%	5.0%	8.2%
Application Category Features	**29.4%**	**34.3%**	**NA**
Switch to/from specific application category (26): switch to/from a specific application category (e.g.,messaging), while the previous one was different. Application categories considered: CodeReview [PS], DeveloperTool [3], IDE [3], Idle [PS],[35], IM [PS],[47], Mail [PS],[47], Music [PS],[15], [22], [47], Navigate, Read/Write Document [3], TestingTool [3], Utility, WebBrowser [PS],[3], Unknown	28.0%	NA	NA
Same application category (1): the current application category is the same as the one in the previous application segment, e.g., both are messaging [15], [47]	1.4%	NA	NA
Time spent per application category (13): the percentage of the total duration of the task segment that was spent in each of the 13 application categories [14], [32], [33]	NA	34.3%	NA
Switching Frequency Features	**16.6%**	**13.7%**	**19.0%**
Difference in the window switches frequency (1): difference of the number of switches between windows of the same or a different application per second between the current and the previous application/task segment [14], [27], [32]	7.2%	13.7%	19.0%
Difference in the time spent in an application (1): difference of the total duration spent between the current and the previous application segment [14]	9.4%	NA	NA
Lexical Features	**8.2%**	**0%**	**0%**
Code in window title (1): the window titles of the current and previous application/task segments both contain code, as identified by text that is written in camelCase or snake_case. Can also distinguish between development and other file types	1.4%	0%	0%
Lexical similarity of the window titles and application names (2): cosine similarity based on the term frequency-inverse document frequency (TF-IDF) between the current and previous application segments' window titles or application names [14], [27], [48]	6.8%	NA	NA

References on these features (in blue) are either on previous related work or participants' suggestions (PS). A feature importance of NA denotes that the feature was not used for the prediction group. For the task type columns, 'All' denotes that all features were considered, 'UI' indicates that only the user interaction features were used, and the application category features were ignored. Numbers in brackets show feature counts.

current with the previous application segments' features, we run the task switch detection taking into account 1 and up to 10 steps back into the past, and comparing the resulting precision and recall. Our analysis of the results indicated that after an initial increase of the precision for detecting a switch, the precision and recall gradually drop as the number of steps increases. We therefore chose 2 as the number of steps to go back in terms of application segments. As a result, the total number of features used for the task switch detection is 84, which is double the number of unique features used: once calculated for comparing the current with the previous application segment, and once to compare the previous two application segments. In the following, we provide an overview over all the features used:

User Input Features. The first feature group are user input features. They are based on keyboard and mouse interaction, such as the difference in the number of keystrokes the participant pressed per second between this and the previous time window segment.

Application Category Features. We categorized commonly used applications into one of 13 predefined application categories, based on our classification in previous work [3] and participants' suggestions of what they consider to be good indicators for switching to another task. These include categories specific to software engineering, such as *DeveloperTool*, *CodeReview* or *TestingTool*, but also more general ones, such as *Read/Write Document*, *Email* and *Web Browser*. They are leveraged in 26 features that capture switches *to or from* a specific application category, such as switching to a messaging application or becoming idle. Since switching to another application might be another indication for a task switch [15], [47], we added one feature that captures these.

Switching Frequency Features. In the post-study questionnaire, participants mentioned that they often navigate through several applications to clean-up their computer right before starting a new task, which is why we added a temporal feature based on the window switching frequency. One feature captures the difference in the time spent in an application, since this might be another indicator for a task switch, either because a switch is less likely immediately after a task switch, and the likelihood of a task switch increases as time passes [14].

Lexical Features. Inspired by prior work [14], [27], [48], we also added three lexical/semantic features that are extracted from application names and their window titles. The textual data was first pre-processed to produce lists of words via tokenization on punctutation and whitespace. From these lists we also removed common stop words such as "and", "the", and "or". Since window titles might include code snippets, such as a class or method name or development file type, we added a feature that captures whether the window title contains text written in `camelCase` or `underscore_case`, and whether this is different to the previous segment. To determine whether the previous and current application segments have a contextual similarity, two features are calculated based on the cosine similarity of the window titles and application names using the term frequency-inverse document frequency (TF-IDF). Note that the application name and window titles were also used to determine the application category features. In addition, and unlike some previous work, we explicitly did not capture file contents to reduce intrusiveness and avoid privacy concerns [37], [51], [52].

4.4 Task Type Features Extracted

For the task type detection, we reused the same features as in the task switch detection whenever possible. However, some features required adaption or made no sense in this context. First, as the time window for task type detection encompasses one or multiple application segments, we replaced the *application category* features with a feature that captures the ratio between the *time spent in the specific application category* and the time spent in the task segment. This allowed us to determine the dominant application category in a task segment. Second, we eliminated the lexical similarity features as these are computed based on an application segment's similarity to another segment. In the task type detection scenario, we have no comparable ground truth to use to calculate such features. This resulted in a total of 25 features used for the task type detection.

4.5 Outcome Measures

For the *task switch detection*, we labeled each application segment either with *Switch* or *NoSwitch*, depending on whether we observed a task switch (in study 1) or whether the participant self-reported a task switch (in study 2). While our model is able to detect task switches on the granularity of application segments, an actual switch might happen while using the same application. Thus, our task switch detection approach is at most the duration of the application segment away from the actual task switch, which was an average of 1.6 minutes ($\pm$2.2) in our study. For the *task type detection*, we labeled each task segment with the observed or self-reported task type. Descriptive statistics regarding participants' task switching behavior and the task types they worked on can be found in Section 5.1 and Section 6.2, respectively.

4.6 Machine Learning Approach

We used scikit-learn [53], a widely used machine learning library for Python, to predict task switches and task types. We evaluated several classifiers by applying them to our feature set and testing different hyperparameters. A RandomForest classifier with 500 estimators outperformed all other approaches, including a Gradient Boost-Classifier, Support Vector Machine (SVM), Neural Network and Hidden Naïve Bayes classifier. Details on the hyperparameters of the evaluated classifiers can be found in the supplementary material [45]. A RandomForest classifier is one form of ensemble learning that creates multiple decision tree classifiers and aggregates their predictions using a voting mechanism [54], [55]. It does not require a pre-selection of features and can handle a large feature space that also contains correlated features. Hence, for the remainder of this paper, the presented results were obtained using a *RandomForest classifier*. Prior to classification, we impute missing values by replacing them with the mean and apply standardization of the features, which centers the data to 0 and scales the standard deviation to 1. These common steps in a machine learning pipeline can improve a classifier's performance [56]. For the *task switch detection*, we further apply Lemaître's implementation of SMOTE, which is a method for oversampling and can considerably boost a classifier's performance in the case of an imbalanced dataset such as ours [57]. For the *task type detection*, where as much as 80-90 percent of the reported types are of the *Development* class we instead employ penalized classification to correct problems caused by class imbalance, as SMOTE has significant drawbacks when the minority classes have a limited number of samples [58].

We built both *individual and general models*, where an individual model is trained and tested with data solely from one participant and a general model is trained on data from all participants except one, and tested on the remaining one. Individual models often have a higher accuracy since they are trained on a person's unique behavioral patterns. On the other hand, general models are usually less accurate but have the advantage of solving the *cold-start-problem*, which means that no prior training phase is required and the model can be applied to new users immediately.

To evaluate the individual models, we applied a 10-fold cross-validation approach, where the model was iteratively tested on 1/10 of the dataset while being trained on the remaining data. We adapted the cross-validation approach to account for the temporal dependency of the samples. In particular, there is a dependency between samples in close temporal proximity, since data from the preceding samples is incorporated in the features. To ensure a valid and realistic evaluation of the model [59], we therefore deleted h samples on either side of the test set block. In our case, we chose h=10 since we included up to 10 preceding samples in the feature calculation (see Section 4.3). The cross-validation approach is illustrated in Fig. 3.

5 Results: Detecting Task Switches

5.1 Descriptive Statistics of the Dataset

Participants switched frequently between tasks, with a mean task switch rate of 6.0 ($\pm$3.7, min: 1.8, max: 18.9) times per hour. The average time spent on each task was 13.2 ($\pm$ 7.3, min: 3.1, max: 30.8) minutes[3]. Developers' task switch behaviors are similar to previous work [2], [23].

3. We do not report individual results for the two studies, since the task switch rate (p-value=.056) and time spent on a task (p-value=.215) are not significantly different in the two datasets.

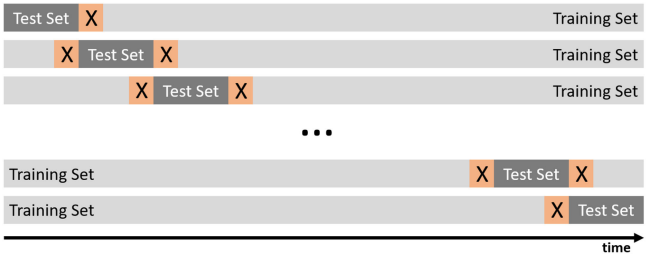

Fig. 3. Cross-validation approach for the individual models, leaving a gap of 10 samples before and after the test set to account for the dependence of samples in close temporal proximity.

5.2 Task Switch Detection Error Distance and Accuracy

To analyze how well our task switch predictions work, we run a first discrete analysis by calculating the *error distance* between each predicted and the actual task switch. The average error distance is 2.79 (±2.30) application-switches, meaning that in case a task switch was *not* detected at the exact moment, it is on average 2.79 applications before or after the predicted one. To put this in context, multiplying the average application segment length of 1.6 (± 2.2) minutes (Section 4.5) with the error distance results in an average of only 4.46 minutes that a task switch is predicted before or after the actual one. Of all task switches that were *not* detected at the exact moment, 44.7 percent of the task switches our model predicted have an error distance of 1 application-switch, 15.8 percent have a distance of 2 application-switches, and 39.5 percent have a distance of 3 or more application-switches.

Table 3 gives an overview of the task switch detection performance of individual and general models. We split the presentation of the data into the two studies, since they were collected with a different method. As a baseline, we report the results of a random classifier, where the likelihood of predicting a certain class is based on the class distribution of the training set.

Overall, our analysis revealed that we can detect task switches at their *exact* location with a high averaged accuracy of 84 percent (precision: 62 percent and recall: 35 percent, kappa: 0.34) when trained with individual models. Applying the general model, we achieved an averaged accuracy of 73 percent as well as higher recall of 55 percent and lower scores in both precision (46 percent) and kappa (0.27). Overall, despite these differences we found the two models were very similar in performance judged by both AUC (74 percent for individual versus 75 percent for general) and F1-score (43 percent versus 40 percent). Compared with our baseline classifier, both the individual and general model show substantial improvements across the board, with the exception of recall in the individual model. Note that this does not mean that the baseline necessarily performed better in this case, only that our model was more much selective in its predictions, as is reflected in the higher precision score. For the individual models, we compared the results of each participant's model (see supplementary material [45]). It reveals that the prediction performance varies quite substantially for each participant. These results are discussed with respect to their applicability in real-world scenarios in Section 7.

5.3 Task Switch Feature Evaluation

A Random Forest classifier can deal well with a larger number of features which makes prior feature dimensionality reduction of our 84 features obsolete [54], [55]. While we do not apply a feature selection technique in our approach since it would only select the most predictive features in the model, we are still interested in learning if certain features are generally more important, especially across different participants. The second column of Table 2 contains the feature importance as attributed by the RandomForest classifier using all features and averaged over all participants' individual models. To calculate the feature importance metrics, we used the Gini impurity measure from scikit-learn, which captures the feature's ability to avoid mis-classification [53]. The most predictive feature groups are user input (45.8 percent) and application category (29.4 percent). The feature group with the least predictive power are the lexical features (8.2 percent). The supplementary material includes the feature importances of each individual feature [45].

6 Results: Detecting Task Types

6.1 Identified Task Type Categories

As described in more detail in Section 3, we inferred task type categories after collecting task and task switch data from observing 12 developers at work and performing a Thematic Analysis. This resulted in nine task type categories we described in Table 4. In the post-study questionnaires of study 2, participants reported that they agreed with the identified task types and generally had no issues to assign them. However, two participants mentioned that a task type for *Support* duties was missing:

> *"[Support]-Duties. These are very specific tasks that require a lot of different things to do. It's not Development and it can be a lot of ad-hoc and requires many context switches."* - P14

TABLE 3
Overview of the Performance of Detecting task Switches, for Both Individual and General Models

	INDIVIDUAL MODELS					GENERAL MODEL				
Dataset	Accuracy	AUC	F-Score	Precision	Recall	Accurracy	AUC	F-Score	Precision	Recall
Study 1: Observations	82%	76%	44%	57%	38%	70%	78%	46%	46%	61%
Study 2: Self-Reports	86%	72%	42%	67%	31%	66%	72%	36%	43%	55%
All	**84%**	**74%**	**43%**	**62%**	**35%**	**67%**	**75%**	**40%**	**46%**	**57%**
Baseline	*55%*	*49%*	*24%*	*18%*	*42%*	*51%*	*50%*	*25%*	*18%*	*48%*

TABLE 4
Overview and Descriptions of the Task Type Categories, the Average Time Developers Spent on Each Task Type Per Hour of Work, and the Performance of our Task Type Detection Approach, for Both Individual and General Models

			INDIVIDUAL MODELS				GENERAL MODEL			
			All Features		UI Features		All Feat.		UI Feat.	
Task Type Category	Avg (Stdev) mins/h	Sample Size	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.
Development: bug-fix, refactoring, code review, implementing new feature, reading/understanding documentation/code, testing, version control, dev.-related learning	37.2 (±12.2)	612	70%	85%	62%	77%	59%	77%	50%	78%
Personal: work unrelated web browsing, private emails or texts, (bio or lunch) break	9.7 (±7.0)	170	48%	45%	42%	34%	33%	32%	32%	23%
Awareness & team: reading/writing emails, discussions/answering questions in IM	5.3 (±6.0)	234	64%	53%	40%	35%	36%	44%	22%	15%
Administrative: often routine tasks, e.g., reporting work-time, expenses report, paperwork	4.0 (±3.6)	12	50%	17%	33%	8%	NA	NA	NA	NA
Planned Meeting: attending a scheduled meeting/call, e.g., weekly scrum, weekly planning meeting	3.6 (±2.7)	94	40%	40%	33%	31%	10%	4%	2%	1%
Unplanned Meeting: attending an ad-hoc, informal meeting, usually with one team-member only, e.g., unscheduled phone call, colleague asking a question	3.1 (±2.8)	90	43%	29%	36%	29%	41%	25%	30%	18%
Planning: in the calendar, task list, work item tracker	3.0 (±3.5)	90	31%	24%	26%	16%	16%	1%	1%	0%
Other: tasks that do not fit into the other categories. Participants mentioned that these were support-duty, document writing (e.g.,in PowerPoint, Word) and for product development/innovation.	2.7 (±3.9)	40	47%	25%	3.7%	2.5%	0%	0%	2%	1%
Study: work related to this study (e.g.,talking to observer, filling out questionnaire)	1.9 (±1.9)	64	67%	58%	49%	41%	78%	69%	29%	20%
All		**1406**	**59%**	**61%**	**46%**	**49%**	**44%**	**50%**	**33%**	**41%**
Baseline		*1406*	*30%*	*30%*	*30%*	*30%*	*24%*	*24%*	*24%*	*24%*

The 'All Features' columns show results using models trained with all features, while the 'UI Features' columns show results from models trained using only user interaction features (i.e., excluding application category features).

Two participants mentioned that it was sometimes difficult to know if time spent on emails should be assigned to *Development* or *Awareness & team*:

> *"I was sometimes unsure of how to classify the time I spent responding to emails. I generally classified it as development since most of the emails were development-related."* - P21

Most of our task type categories are consistent with previous work that investigated knowledge workers' tasks [32], [60], [61], [62]. For example, *Meetings*, *Administrative*, *Planning* and *Private* were also prevalent in both Kim *et al.*'s and Czerwinksi *et al.*'s work [60], [62]. Kim *et al.* further divided project work (in our case *Development* tasks) into *Documenting and Conceptualizing Ideas*, *Environment and Development* and *Design*. We did not make these finer-granular distinctions since we did not want to make the self-reporting of task types in the second study too complicated, which would degrade the quality of self-reports.

6.2 Descriptive Statistics of the Dataset

On average, developers worked on 6.1 (±1.6, min: 3, max: 9) different task types during the studied time periods, indicating that most of the identified task types are relevant to all developers. The majority of developers worked on *Development*, *Awareness & team*, *Personal* and *Planning tasks* on a daily basis. Only five developers worked on *Administrative* tasks during the study period, indicating that for many developers this is not a task they spend time on very often. The task type participants self-reported having spent the most time on is *Development*, with an average of 37 (± 12) minutes spent for every hour of work. Participants also spent a surprisingly high amount of time, almost 10 minutes per hour of work, with *Personal*, including work unrelated browsing and messaging. Table 4 reports details for all task types as well as the number of participants who self-reported having worked on the task type.

We also analyzed if having a higher diversity in work (i.e., working on more different task types) correlates with developers switching more between tasks. There is a weak, not statistically significant positive correlation (Pearson's $r = 0.32, p = .12$), which suggests that there are other, more important reasons causing developers to switch tasks.

6.3 Task Type Detection Accuracy

Table 4 shows the results of our task type detection approach across all 9 task type categories. We omit the accuracy metric in this table, as recall is a measure of individual class accuracy, and since the recall presented in the *all* row is weighted by class size it therefore assumes exactly the same value as accuracy. As with the task switch detection analysis, we trained both individual models and one general model which was trained on all participants. The *Administrative* task type was not predicted a single time by the general classifier, and as thus the precision scores were undefined for this class. Similarly, the task types *Planning* and *Other* had low precision and recall values, since the

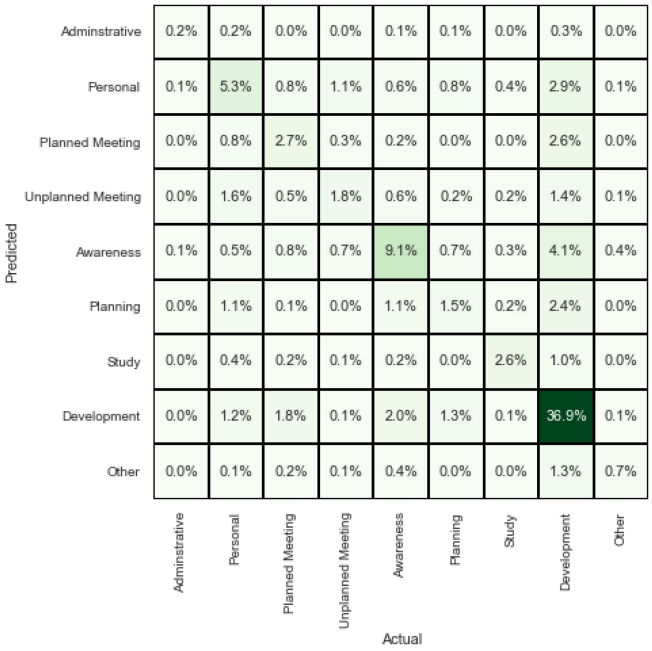

Fig. 4. Confusion matrix for task type prediction.

sample size used for training these types was small. In general, the individual models (precision 59 percent, recall 61 percent) outperformed the general model by a large margin (precision 44 percent, recall 50 percent).

One important aspect of our approach that distinguishes our classifier from previous work (e.g., [15], [22], [47]) is its ability to make predictions even on previously unseen applications. To demonstrate this, we split the results into two categories: with the manual application category mappings (*All Features*) and without (*UI Features*). The *UI Features* include all user interaction features, but exclude application features. While the combined approach proved to be superior, user input features still proved to have high predictive power on their own. Overall, there was a 28.2 percent increase in precision when including the application category features, and a 24.5 percent increase in recall.

We also found there was a substantial difference in performance depending on the task type category. The *Development* task type proved to be the easiest to predict, achieving high recall (85 percent) and precision (70 percent) scores. Conversely, the *Planning* task type saw very poor results, with only 24 percent recall and 31 percent precision. These results are somewhat in line with what one might expect. Naturally, some task categories are more difficult to predict than others. For instance, discerning the nature of a meeting (planned or unplanned) based purely on a users applications used and input activity seems to be nearly impossible. As seen in Fig. 4, there is substantial confusion between some categories, especially between the two meeting categories (*Planned Meeting* and *Unplanned Meeting*) and the *Personal* category. These categories tended to have a high amount of time spent idle, meaning the participant was away from their computer which naturally makes correct predictions exceptionally difficult. As a consequence of the dominance of *Development* samples in our dataset, our classifier also exhibits a strong bias towards predicting the *Development* category. While a larger sample size would likely help reduce this bias, it is of note that the *Development* category is also the one participants spent the majority of their time in, 37.2 ($\pm$12.2) minutes on average for every hour of work.

6.4 Task Type Feature Evaluation

The third and fourth column of Table 2 show the Gini feature importances we calculated for our RandomForest classifiers, averaged over all participants. When considering *All Features*, we found the *time spent per application category* features to have by far the greatest importance (39.1 percent), followed by keystroke features (17.5 percent). However, the combined *user input feature group* contributed more than any other feature group (47.6 percent). The lexical features did not contribute at all to the results of the classifier, which suggests there is room for improvement in this area as window titles can contain a substantial amount of hints that could help to identify a specific task. The supplementary material includes individual task type feature importances.

7 Discussion

In this section, we discuss implications of our results, possible improvements to automated task switch and type detection, and practical applications of automated task detection in real-world scenarios.

7.1 Improving Task Switch Detection

We found that for the task switch detection, the individual models perform quite *similarly* to the general model overall, even though the prediction performance varies quite substantially for each participant. This suggests that using the general classifier is accurate enough to solve the cold-start problem. For practical, real-world applications, we therefore suggest using a general model as a default, and then allowing the user to improve the classifier by training it. As we found in study 2, collecting periodic self-reports over just a few days is feasible in real-world scenarios and may even lead to some insights about work itself.

More research is required to explore reasons for and better balance the individual differences in developers' task switching behaviors. This includes investigating the characteristics of inaccurately classified task switches and consider additional data sources. For example, we could imagine to include information about a developer's personality and company culture to train a classifier that works well for developers with similar work habits, instead of building a general one for everyone. Future work could also study the predictive power of features extracted from additional data sources, such as emails, calendars, biometrics (e.g.,detecting when a user is away from the computer), and more detailed development related data (e.g.,activities inside the IDE).

The relatively low feature importances of our lexical features shows further potential to more effectively leverage contextual information. Besides calculating lexical similarity based on cosine similarity (TF-IDF) of window titles, we also experimented with variations, such as an unweighted term frequency metric and two different word embedding models including one trained on Wikipedia, and one trained on

StackOverflow which has been shown to produce embeddings that are more closely related to the domain of software engineering[63]. They led to even less predictive features, which is why we did not report them separately. One reason could be the little overlap in the window title data. Window titles generally capture only the application name and the name of the current file, document, email or website, which limits overlaps with other window titles. Including the actual contents of these resources could be one way to overcome these limitations, but could result in privacy concerns, as discussed in more detail in the next section.

7.2 Improving Task Type Detection

For the task type predictions we found that the individual models *outperform* the general model, with an overall accuracy of 61 percent compared to 50 percent. Even though we collected data from a rather large sample of 25 participants (compared to similar work), we were not yet able to build highly reliable general models, which could solve the cold-start problem. The difficulty to discover common patterns across all participants emphasizes how individual and diverse developers' tasks are.

We see our work as a first *step* towards better understanding and *automatically* characterizing developers' task context. With our models' ability to automatically detect task switches based on data collected through our computer interaction monitoring, a next step could be to collect a more in-depth set of data in-between two task switches, and from more participants over a longer period of time. For example, IDE extensions (e.g., Feedbag [64] or WatchDog [65]) could be leveraged to identify the code files, code reviews, and projects the developer has been working on, browser trackers (e.g., RescueTime [66]) could identify and classify the websites a developer visited, and integrations into the email or IM client could help to understand which people a developer communicated with. To better manage these large amounts of data, research will need to come up with approaches to model and summarize task context—and task types are a first step into doing that. We have been able to find only very little work on *automatically* detecting, characterizing and summarizing (developers) tasks yet [22], [32], [33].

While more fine-grained lexical data, such as the file or website contents (as applied in [37], [51], [52]) or participants' actual keyboard input, could be leveraged to improve our models, it also might reveal details about the company's products or the developers' work and personal life that they are not comfortable sharing with us. To minimize privacy concerns, we had to find a trade-off between *intrusiveness*, by capturing only a minimum set of data, and *completeness*, by monitoring as much as possible to get enough data that allowed us to predict task switches and types in the field. To earn participants' trust with capturing potentially sensitive data, we were also very transparent with what data we collect and how it will be used, allowed participants to review it before sharing it with us, and making it possible to pause the monitoring application at any time that seemed particularly sensitive to them.

7.3 Reducing the Prediction Delay

Ideally, a task switch and type detection would be very close to real-time, i.e., close to the exact time a switch occurs. With our approach, there can be a *prediction delay* of a maximum of one unique application segment, on average 1.6 minutes ($\pm$2.2), when predicting a task switch. This delay is considerably smaller compared to previous approaches that applied fixed window lengths of (usually) 5 minutes (e.g., [13], [14], [17], [27], [32], [33]). Nonetheless, future work could further reduce the prediction delay by further shortening the smallest possible segment size, in our case application switches. This would allow to also identify switches within an application, such as when a developer is switching tasks inside the web browser or IDE.

7.4 Applications for Automated Task Detection

An active area of research aims to better support developers' frequent task switching, for example by supporting resuming interrupted tasks or by easing task switching (see Section 2.3). So far, most approaches are limited to developers' *manual* identification of task switches, and their evaluations have pointed out challenges this poses for them. Our approach demonstrates the feasibility of *automatically* detecting task switches and types in the field, based on a few hours of training data, which makes it possible to increase the value of previous approaches significantly and stimulate new research and tool support. Notably, tool support would greatly benefit from the improvements we discussed in the sections above. In the post-study questionnaire of study 2, participants described concrete applications that we qualitatively analyzed and related to prior work, which resulted in the following three main opportunities for applying automated task detection:

One application of an almost real-time detection of task switches that 8 (out of 13) participants described is to *actively reduce task switching*. This includes automatically blocking notifications from email, instant messaging or social networks when a developer is focused on a (challenging) task, to allow extended times of deep focus:

> *"What if Windows has a built-in and personalized model about when to give you notifications. I feel like there is a good middle ground between forcing the user to turn off notifications from the OS and having too many notifications interrupting the user."* - P25

Reducing task switching at times of high focus could greatly reduce multi-tasking, a major source of stress and quality issues [2], [67], [68], [69]. Similarly, an automated task switch detection could improve interruptibility classifiers and postpone in-person interruptions from co-workers to task switch borders, times they are less costly [70], [71], [72].

Another application of automated task detection could be to support the *resumption of suspended or interrupted tasks*. Participants did not suggest this application themselves, but 8 (out of 13) rated it as 'useful' or 'very useful' in a follow-up question of the final questionnaire. According to Parnin and Rugaber, a major challenge of task resumption is to rebuild the interrupted task's context [73]. Applying similar summarization approaches as seen in other areas of software development [74], [75] could be presented to the user as cues upon returning to the suspended task, which has been shown to considerably reduce the resumption

lag [4], [76], [77]. While previous approaches, such as TaskTracer [7], Scalable Fabric [78], GroupBar [9] and Mylyn [6], allow the capturing and presentation of task context, they require the user to *manually* group related artifacts or manually state the start and end of a task, thus, reducing chances of long-term adoption. Even tough there is room for improvement as discussed above, our approach can serve as a starting point to automate these approaches, since it can already be beneficial to receive help with resuming some tasks, as long as they are detected correctly.

A third opportunity of application that 10 (out of 13) participants suggested is to use *automated* task detection to *increase their awareness about task work* and time spent on tasks, which could help to identify opportunities for work habit and productivity improvements. This is in line with a survey with 379 developers that showed the most-often mentioned measurement of interest when reflecting about productivity are the tasks developers made progress on and completed in a workday [23]. An aggregated visualization of the automatically inferred tasks could give developers insights such as how much time they spend on different tasks, when they worked on planned versus unplanned tasks, or their multi-tasking behaviors:

> *"It can help point out different working styles that are also effective and efficient. Not everyone works in the same way."* - P 24

Recently, researchers started building retrospective dashboards for developers [42], [65], [79], [80] and other knowledge workers [66], [81], [82], usually by visualizing data on the level of applications or application categories, but suggesting that a per-task level would be more beneficial. An increased awareness about one's task switching behavior could support developers to identify goals that help to maintain and improve good work habits, such as reducing multi-tasking or actively blocking notifications from distracting services and websites at times they need to focus. Participants further suggested that the data could help to reduce administrative workloads that require them to report time spent at work:

> *"We're often asked to report at the end of the month how much time we spent on support requests (...) versus development work. That kind of info is tedious to track manually, but a tool could generate an automatic report as needed, allowing for more accurate counts."* - P22

Lu *et al.* recently showed that the lack of logs of activities and tasks is often a hindrance to be able to transfer them into time reports [83]. While a few time-tracking tools already exist (e.g.,DeskTime [84], TimeDoctor [85]), they all require users to manually specify the start and end of a task.

8 Threats to Validity

Observing Developers in Study 1. The internal validity of our results might be threatened by the presence of the observers during the observation sessions, causing developers to diverge from their regular work habits, e.g., having less breaks than usual. Observing participants on a single day only might not be representative of the participant's regular workday. We tried to mitigate these risks by not interacting with participants during the observations, splitting up the session into two two-hour blocks, sitting as far away from the participant as possible, telling co-workers beforehand that they could still communicate and interrupt as usual, and by allowing the participant to pick an optimal timeslot that is representative of their usual work. Our observational study has the advantage that, rather than performing a lab study or experimental exercise, participants were observed during their real-world work, thus increasing generalizability and realism. However, the above mentioned risks of observing developers at their workplace make it very difficult to scale observational studies and observe them over many days. Hence, we did not rely only on observations, but also on participants' self-reports and with that, combining two methods and strengthening our overall approach.

Self-Reporting in Study 2. While collecting participants' task data using self-reports has proven to be a valuable approach to scale the collection of labeled data for supervised learning, there are a few limitations. First, we rely on the accuracy of participants' self-reports. For example, they might not always have been able to accurately remember their tasks, or filling out the pop-up regularly might be perceived as cumbersome after a while. In Section 3.2, we describe our actions to minimize these risks in detail, including the ability to postpone a pop-up and collecting confidence ratings. Aiming to make the self-reporting as easy as possible required limiting the self-reports to segments with the granularity of an application switch and excluding application switches shorter than 10 seconds. This is why our models are unable to detect task switches within an application, as well as very short ones. Since developers switch between applications very frequently, on average every 1.6 minutes ($\pm$2.2), our model is able to predict a task switch within the same time frame. Future work could investigate how to give participants good-enough cues that allow them to accurately self-report switches within applications (e.g.,switching from a news website to the work item tracker in the browser) without making the interface too cluttered. Finally, the reliance on collecting computer interaction data only, instead of also including other sensors such as heart-rate monitors or cameras, limits our knowledge of what is happening when there is no input to the computer, e.g., in the case of idle times from using the smartphone, reading a document without scrolling, or a discussion with a co-worker.

Sample Size. A further threat to the external validity of our results could be the number of participants. A higher number of participants might have led to a more robust general model to predict task switches and task types. Nonetheless, collecting task data from 25 participants is considerably higher than what was reported in previous work (between 1 and 11 participants). We tried to mitigate this threat by selecting participants from four different software companies in various locations.

Task Definitions. The construct validity of our results might be threatened by our definitions of a task (switch) and our open coding approach to identify task type categories. To minimize this risk, we based our definitions of task, task switch and task type on previous work, and asked participants about their own definitions in both studies (Section 3).

9 Conclusion

In this paper, we explored the potential of *automatically* detecting task switches and task types based on developers' computer interaction data. Running two field studies that we conducted with a total of 25 professional software developers, we found that we are able to detect task switches and task types with high accuracy in the field and within a short time frame (average 1.6 minutes) from the actual task switch. We thereby examined a broad range of semantic and temporal features extracted from the computer interaction data and found that features based on user input data hold the highest predictive power.

Our work extends previous work with an approach that uses a broader range of temporal and semantic features, by developing new features, and by not being limited to capturing task switches and types within the IDE only. The evaluation of our approach in a field-study with 25 professional developers, compared to 1 to 11 participants in previous work, revealed higher accuracy and less delay in the predictions than comparable prior work. The strong evidence on the potential to automatically predict task switches in the field opens up a wide range of applications in real-world work settings, ranging from complementing existing *manual* task support, such as Mylyn [6], to automating time tracking tools, all the way to new tool support to leverage developers' workflows.

References

[1] D. E. Perry, N. Staudenmayer, and L. G. Votta, "People, organizations, and process improvement," *IEEE Softw.*, vol. 11, no. 4, pp. 36–45, Jul. 1994.

[2] V. M. González and G. Mark, ""Constant, constant, multi-tasking craziness": Managing multiple working spheres," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2004, pp. 113–120.

[3] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz, "The work life of developers: Activities, switches, and perceived productivity," *IEEE Trans. Softw. Eng.*, vol. 43, no. 12, pp. 1178–1193, Dec. 2017.

[4] 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.

[5] G. C. Murphy, M. Kersten, M. P. Robillard, and D. Čubranić, "The emergent structure of development tasks," in *ECOOP 2005 - Object-Oriented Programming*. Berlin, Germany: Springer, 2005, pp. 33–48.

[6] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," in *Proc. 14th ACM SIGSOFT Int. Symp. Foundations Softw. Eng.*, 2006, pp. 1–11.

[7] A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker, "Tasktracer: A desktop environment to support multi-tasking knowledge workers," in *Proc. 10th Int. Conf. Intell. User Interfaces*, 2005, pp. 75–82.

[8] A. Sahm and W. Maalej, "Switch! recommending artifacts needed next based on personal and shared context," in *Proc. Softw. Eng. Workshops*, 2010, pp. 473–484.

[9] G. Smith *et al.*, "Groupbar: The taskbar evolved," in *Proc. OZCHI*, 2003, vol. 3, pp. 1–10.

[10] S. K. Card and A. Henderson Jr, "A multiple, virtual-workspace interface to support user task switching," *ACM SIGCHI Bull.*, vol. 17, no. SI, pp. 53–59, 1986.

[11] A. Bragdon *et al.*, "Code bubbles: Rethinking the user interface paradigm of integrated development environments," in *Proc. 32Nd ACM/IEEE Int. Conf. Softw. Eng.* - Vol. 1, 2010, pp. 455–464.

[12] A. Bragdon *et al.*, "Code bubbles: A working set-based interface for code understanding and maintenance," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.* 2010, pp. 2503–2512.

[13] J. Shen, L. Li, and T. G. Dietterich, "Real-time detection of task switches of desktop users," in *Proc. IJCAI*, 2007, vol. 7, pp. 2868–2873.

[14] J. Shen *et al.*, "Detecting and correcting user activity switches: Algorithms and interfaces," *Proc. 14th Int. Conf. Intell. User Interfaces*, 2009, pp. 117–126.

[15] H. T. Mirza, L. Chen, G. Chen, I. Hussain, and X. He, "Switch detector: An activity spotting system for desktop," in *Proc. 20th ACM Int. Conf. Inform. Knowl. Manage.*, 2011, pp. 2285–2288.

[16] S. Stumpf *et al.*, "Predicting user tasks: I know what you're doing," in *Proc. 20th Nat. Conf. Artif. Intell. Workshop Hum. Comprehensible Mach. Learn.*, 2005.

[17] R. Nair, S. Voida, and E. D. Mynatt, "Frequency-based detection of task switches," in *Proc. 19th Brit. HCI Group Annu. Conf.*, 2005, vol. 2, pp. 94–99.

[18] J. Shen, L. Li, T. G. Dietterich, and J. L. Herlocker, "A hybrid learning system for recognizing user tasks from desktop activities and email messages," in *Proc. 11th Int. Conf. Intell. User Interfaces*, 2006, pp. 86–92.

[19] M. K. Gonçalves, L. 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.

[20] 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.

[21] S. Astromskis, G. Bavota, A. Janes, B. Russo, and M. D. Penta, "Patterns of developers' behaviour: A 1000-hour industrial study," *J. Syst. Softw.*, vol. 132, pp. 85–97, 2017.

[22] L. Bao, Z. Xing, X. Xia, D. Lo, and A. E. Hassan, "Inference of development activities from interaction with uninstrumented applications," *Empir. Softw. Eng.*, vol. 23, no. 3, pp. 1313–1351, 2018.

[23] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann, "Software developers' Perceptions of productivity," in *Proc. 22nd ACM SIGSOFT Int. Symp. Foundations Softw. Eng.*, 2014, pp. 19–29.

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

[25] H. T. Mirza, L. Chen, A. Majid, and G. Chen, "Building user task space by mining temporally proximate desktop actions," *Cybern. Syst.*, vol. 42, no. 8, pp. 585–604, 2011.

[26] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering Using UML, Patterns and Java-(Required)*. Englewood Cliffs, NJ, USA: Prentice Hall, 2004, vol. 2004.

[27] N. Oliver, G. Smith, C. Thakkar, and A. C. Surendran, "Swish: Semantic analysis of window titles and switching history," in *Proc. 11th Int. Conf. Intell. User Interfaces*, 2006, pp. 194–201.

[28] M. Robillard and G. Murphy, "Program navigation analysis to support task-aware software development environments," *Proc. ICSE*, 2004, pp. 83–88.

[29] I. D. Coman and A. Sillitti, "Automated identification of tasks in development sessions," in *Proc. 16th IEEE Int. Conf. Program Comprehension*, 2008, pp. 212–217.

[30] L. Zou and M. W. Godfrey, "An industrial case study of coman's automated task detection algorithm: What worked, what didn't, and why," in *Proc. 28th IEEE Int. Conf. Softw. Maintenance*, 2012, pp. 6–14.

[31] K. Kevic and T. Fritz, "Towards activity-aware tool support for change tasks," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2017, pp. 171–182.

[32] S. Koldijk, M. van Staalduinen, M. Neerincx, and W. Kraaij, "Real-time task recognition based on knowledge workers' computer activities," in *Proc. 30th Eur. Conf. Cognitive Ergonom.*, 2012, pp. 152–159.

[33] H. T. Mirza, L. Chen, I. Hussain, A. Majid, and G. Chen, "A study on automatic classification of users' desktop interactions," *Cybern. Syst.*, vol. 46, no. 5, pp. 320–341, 2015.

[34] M. Hassib, M. Khamis, S. Friedl, S. Schneegass, and F. Alt, "Brainatwork: Logging cognitive engagement and tasks in the workplace using electroencephalography," in *Proc. 16th Int. Conf. Mobile Ubiquitous Multimedia*, 2017, pp. 305–310.

[35] W. Maalej, M. Ellmann, and R. Robbes, "Using contexts similarity to predict relationships between tasks," *J. Syst. Softw.*, vol. 128, pp. 267–284, 2017.

[36] G. Robertson *et al.*, "The task gallery: A 3D window manager," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2000, pp. 494–501.

[37] T. Rattenbury and J. Canny, "Caad: An automatic task support system," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2007, pp. 687–696.
[38] N. Oliver, M. Czerwinski, G. Smith, and K. Roomp, "Relalttab: Assisting users in switching windows," in *Proc. IUI*, 2008, pp. 385–388.
[39] I. Safer and G. C. Murphy, "Comparing episodic and semantic interfaces for task boundary identification," in *Proc. Conf. Center Adv. Stud. Collaborative Res.*, 2007, pp. 229–243.
[40] C. Parnin and R. DeLine, "Evaluating cues for resuming interrupted programming tasks," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2010, pp. 93–102.
[41] H. Mintzberg, *The Nature of Managerial Work*, ser. Theory of management policy series, Englewood Cliffs, NJ, USA: Prentice-Hall, 1980.
[42] A. N. Meyer, G. C. Murphy, T. Fritz, and B. Columbia, "Design recommendations for self-monitoring in the workplace: Studies in software development," *CSCW*, vol. 1, no. 1, 2017, pp. 1–24.
[43] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Res. Psychol.*, vol. 3, pp. 77–101, Jan. 2006.
[44] R. Tourangeau, L. J. Rips, and K. Rasinski, *The Psychology of Survey Response*. Cambridge, UK: Cambridge Univ. Press, 2000.
[45] A. N. Meyer *et al.*., supplementary Material, 2020. [Online]. Available: https://doi.org/10.5281/zenodo.2574049
[46] S. T. Iqbal and B. P. Bailey, "Understanding and developing models for detecting and differentiating breakpoints during interactive tasks," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2007, pp. 697–706.
[47] S. T. Iqbal and B. P. Bailey, "Effects of intelligent notification management on users and their tasks," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2008, pp. 93–102.
[48] O. Brdiczka, N. M. Su, and J. B. Begole, "Temporal task footprinting: Identifying routine tasks by their temporal patterns," in *Proc. 15th Int. Conf. Intell. User Interfaces*, 2010, pp. 281–284.
[49] M. Basseville *et al.*, *Detection of Abrupt Changes: Theory and Application*. Englewood Cliffs, NJ, USA: Prentice Hall, 1993, vol. 104.
[50] F. Gustafsson and F. Gustafsson, *Adaptive Filtering and Change Detection*. Citeseer, 2000, vol. 1.
[51] I.-C. Wu, D.-R. Liu, and W.-H. Chen, "Task-stage knowledge support: Coupling user information needs with stage identification," in *Proc. IEEE Int. Conf. Inform. Reuse Integr., Conf.*, 2005, pp. 19–24.
[52] C. A. N. Soules and G. R. Ganger, "Connections: Using context to enhance file search," in *Proc. 20th ACM Symp. Operating Syst. Princ.*, 2005, pp. 119–132.
[53] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learning Res.*, vol. 12, pp. 2825–2830, 2011.
[54] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[55] A. Liaw *et al.*, "Classification and regression by randomforest," *R News*, vol. 2, no. 3, pp. 18–22, 2002.
[56] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, no. Oct, pp. 2825–2830, 2011.
[57] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *J. Mach. Learn. Res.*, vol. 18, no. 17, pp. 1–5, 2017.
[58] G. H. Nguyen, A. Bouzerdoum, and S. L. Phung, "Learning pattern classification tasks with imbalanced data sets," *Pattern Recognition*, Oct. 2009.
[59] J. Racine, "Consistent cross-validatory model-selection for dependent data: Hv-block cross-validation," *J. Econom.*, vol. 99, no. 1, pp. 39–61, 2000.
[60] M. Czerwinski, E. Horvitz, and S. Wilhite, "A diary study of task switching and interruptions," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2004, pp. 175–182.
[61] W. Reinhardt, B. Schmidt, P. Sloep, and H. Drachsler, "Knowledge worker roles and actions - results of two empirical studies," *Knowl. Process Manage.*, vol. 18, no. 3, pp. 150–174, 2011.
[62] Y.-H. Kim and E. K. Choe, "Understanding personal productivity: How knowledge workers define, evaluate, and reflect on their productivity," in *Proc. CHI Conf. Hum. Factors Comput. Syst.* , 2019, pp. 1–2.
[63] V. Efstathiou, C. Chatzilenas, and D. Spinellis, "Word embeddings for the software engineering domain," in *Proc. 15th Int. Conf. Mining Softw. Repositories*, 2018, pp. 38–41.
[64] S. Amann, S. Proksch, and S. Nadi, "Feedbag: An interaction tracker for visual studio," in *Proc. IEEE 24th Int. Conf. Program Comprehension*, 2016, pp. 1–3.
[65] M. Beller, I. Levaja, A. Panichella, G. Gousios, and A. Zaidman, "How to catch 'em all: Watchdog, a family of ide plug-ins to assess testing," in *Proc. IEEE/ACM 3rd Int. Workshop Softw. Eng. Res. Ind. Practice*, 2016, pp. 53–56.
[66] RescueTime, retrieved March 19, 2020, 2020. [Online]. Available: https://rescuetime.com
[67] G. Mark, D. Gudith, and U. Klocke, "The cost of interrupted work: More speed and stress," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2008, pp. 107–110.
[68] G. Mark, S. Iqbal, and M. Czerwinski, "How blocking distractions affects workplace focus and productivity," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput., Proc. ACM Int. Symp. Wearable Comput. - UbiComp* , 2017, pp. 928–934.
[69] G. Mark, M. Czerwinski, and S. T. Iqbal, "Effects of individual differences in blocking workplace distractions," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2018, pp. 1–12.
[70] J. Fogarty, A. J. Ko, H. H. Aung, E. Golden, K. P. Tang, and S. E. Hudson, "Examining task engagement in sensor-based statistical models of human interruptibility," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2005, Art. no. 331.
[71] M. Züger *et al.*, "Reducing interruptions at work: A large-scale field study of FlowLight," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2017, pp. 61–72.
[72] M. Züger, S. C. Müller, A. N. Meyer, and T. Fritz, "Sensing interruptibility in the office: A field study on the use of biometric and computer interaction sensors," in *Proc, CHI Conf. Hum. Factors Comput. Syst.*, 2018, pp. 1–14.
[73] C. Parnin and S. Rugaber, "Resumption strategies for interrupted programming tasks," in *Proc. IEEE 17th Int. Conf. Program Comprehension*, 2009, pp. 80–89.
[74] C. Treude, F. Figueira Filho, and U. Kulesza, "Summarizing and measuring development activity," in *Proc. 10th Joint Meet. Foundations Softw. Eng.*, 2015, pp. 625–636.
[75] N. Nazar, Y. Hu, and H. Jiang, "Summarizing software artifacts: A literature review," *J. Comput. Sci. Technol.*, vol. 31, no. 5, pp. 883–909, Sep. 2016.
[76] A. Rule, A. Tabard, K. Boyd, and J. Hollan, "Restoring the context of interrupted work with desktop thumbnails," in *Proc. 37th Annu. Meet. Cognitive Sci. Soc.*, 2015, pp. 1–6.
[77] E. M. Altmann and J. G. Trafton, "Task interruption: Resumption lag and the role of cues," 2004.
[78] G. Robertson *et al.*, "Scalable fabric: Flexible task management," in *Proc. Work. Conf. Adv. Vis. Interfaces*, 2004, pp. 85–89.
[79] Codealike, retrieved March 19, 2020, 2020. [Online]. Available: https://codealike.com
[80] Wakatime, retrieved March 19, 2020. 2020. [Online]. Available: http://wakatime.com
[81] 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. Hum. Factors Comput. Syst.* , 2016, pp. 272–283.
[82] S. Whittaker, V. Hollis, and A. Guydish, "Don't waste my time: Use of time information improves focus," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2016, pp. 1729–1738.
[83] D. Lu, J. Marlow, R. Kocielnik, and D. Avrahami, "Challenges and opportunities for technology-supported activity reporting in the workplace," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2018, pp. 170:1–170:12.
[84] DeskTime, retrieved March 19, 2020, 2020. [Online]. Available: https://desktime.com
[85] TimeDoctor, retrieved March 19, 2020, 2020. [Online]. Available: https://timedoctor.com

André N. Meyer received the PhD degree in somputer science from the University of Zurich, Switzerland, supervised by Professor Thomas Fritz. His research interests lie in developers' productivity and work, and in creating tools that foster productive work by using persuasive technologies such as self-monitoring, self-reflection and goal-setting. He also works in the information technology industry as an application developer and consultant. For more information, please visit https://andre-meyer.ch.

Chris Satterfield is currently working toward the MSc degree in the University of British Columbia, co-supervised by Dr. Gail C. Murphy and Dr. Thomas Fritz. His research interests are in software engineering, specifically in investigating how machine learning powered tools can be used to improve developer productivity. He is also interested in how biometric monitoring can improve our understanding of a developers emotional state in real time, and how that may impact productivity.

Manuela Züger received the PhD degree in computer science from the University of Zurich, Switzerland, supervised by Professor Thomas Fritz. She is a consultant in the Swiss information technology industry. She has a strong interest in how knowledge workers (in particular, software developers) can be supported in handling interruptions at the workplace by measuring their flow state and developing tool support to reduce costly interruptions.

Katja Kevic received the PhD degree from the University of Zurich. She is a software engineer with Microsoft, UK. Her research interests include various aspects around continuously deploying software.

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 of computer science and associate dean (Research and Graduate Programs) with the Faculty of Science at the University of British Columbia. She is also a co-founder and chief scientist at 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, in 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. For more information, please visit http://thomas-zimmermann.com.

Thomas Fritz received the Diploma degree from the Ludwig-Maximilians-University Munich, Germany, in 2005, and the PhD degree from the University of British Columbia, Canada. He is a professor with the Department of Informatics at the 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.