Master

November 8, 2018

Software Developers' Desktop Interactions

An Analysis

Raphael Imanuel Rosenast

of Kirchberg SG, Switzerland (10-924-728)

supervised by

Prof. Dr. Thomas Fritz

University of Zurich^UZH

s.e.a.l. software evolution & architecture lab

Master

Software Developers' Desktop Interactions

An Analysis

Raphael Imanuel Rosenast

University of Zurich^UZH

s.e.a.l. software evolution & architecture lab

Master

Author: Raphael Imanuel Rosenast, raphael.rosenast@uzh.ch

Project period: 05/08/2018 - 11/08/2018

Software Evolution & Architecture Lab
Department of Informatics, University of Zurich

Acknowledgements

I would like to take this opportunity to thank all those who have accompanied and supported me during the pursuit of this masters thesis.

First of all, my supervisor Prof. Dr. Thomas Fritz has earned a very special thank you. In the course of this work, he has faithfully assisted and supported me in all situations. Thank you for your patience and effort.

Next, I would like to thank the people in the team of Prof. Fritz and in the software and architecture lab of the University of Zurich, that were of great support:
Thank you, Nick Bradley for your open ear, your valuable comments and suggestions in the development and analysis, as well as for the meticulous proof reading of the work.
Thank you, André Meyer for your friendly support and ideas in the development, along with the late night builds and deployment of the application.

In the following I would like to thank the three software companies, and especially the people behind them, that enabled the study trough their time, effort and participation.
I would like to thank the Vertec AG, Zurich, namely I would like to thank Samuel Iseli who personally taught me a lot, who always created an enjoyable work atmosphere and allowed for a great start of the study.
I also would like to express many thankfulness to the Zeit AG, Sursee. I owe special gratitude to Tim Kleinholz who invited us in and started the study with the company.
Further appreciation goes to Logitech Europe S.A., Lausanne and especially Benjamin Vullioud for having an open door and enabling the study as well as Ludovico Novelli for supporting the installation and setup process.
Moreover, I would like to thank all individual participants of the study. Thank you very much for your time and participation.

Certainly not the last thank you goes to my parents and my girlfriend, without whose support this work would have probably never come about.

Abstract

Software development is a complex task and developers need to use a variety of applications for their daily work. The demand to transfer information makes it necessary to use many of the applications simultaneously. This often results in a myriad of open windows, which may reduce navigation efficiency and focus as the number of windows increases. In this thesis, we analyzed how professional software developers interact with and manage their desktop environment. In particular, we look to see if developer efficiency could be affected by high numbers of opened windows. To construct the dataset, we observed the desktop environments of 12 professional software developers from three different companies over a combined total of 195 days. For this task, we used a monitoring application also capturing visual focus with an eye tracker. The eye tracker provided valuable insights additional to the traditional interaction data. We found only 79% of the visual attention was directed at the window with the keyboard input. Half of the desktop environments had 10 or more windows open while mostly only two were fully visible. The number of open windows grows over the course of a work day and we learned most developers do not proactively close windows. From time to time, we could observe desktop environments go through cleanup cycles. Nonetheless, found evidence of unused windows overcrowding the desktop environments and see potential to foster developers focus in the future.

Zusammenfassung

Softwareentwicklung ist eine komplexe Aufgabe und Entwickler benötigen zahllose Applikationen für ihre Tätigkeit. Mit der Anforderung Informationen zu transferieren entsteht die Notwendigkeit diese Applikationen synchron zu verwenden. Resultat sind oft unzählige geöffnete Fenster, welche die Navigationseffizienz mindern und den Fokus beeinträchtigen. In dieser Arbeit wird analysiert wie professionelle Softwareentwickler mit ihrer Desktop-Umgebung interagieren. Besonders wird Effizienz in Zusammenhang mit vielen geöffneten Fenstern thematisiert. Dazu wurde eine Überwachungssoftware entwickelt, welche in der Lage ist Informationen zu Desktop-Umgebung und visuellem Fokus zu erfassen. Die Software wurde bei einem Dutzend professioneller Softwareentwickler von drei verschiedenen Firmen eingesetzt. Dabei wurden Daten im Gesamtumfang von 195 Tagen gesammelt. Im Vergleich zu herkömmlichen Interaktionsdaten konnten mit dem Eye Tracker wertvolle Zusatzinformationen gewonnen werden. Nur 79% der erfassten Blicke waren auf aktive Fenster gerichtet. Auf mehr als der Hälfte der gesammelten Desktop-Umgebungen waren 10 und mehr Fenster gelichzeitig geöffnet. Zudem ist die Anzahl der geöffneten Fenster über den Arbeitstag kontinuierlich gewachsen. Die meisten Entwickler gaben an, Fenster nicht vorausschauend zu schliessen. Gelegentlich wurde die Desktop-Umgebung aufgeräumnt und damit geöffnete Fenster geschlossen. Nichtsdestotrotz haben wir in vielen Fällen Indizien für unbenutzte Fenster gefunden und sehen Potenzial, Entwickler zukünftig in ihrem Fokus zu unterstützen.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The work of a software developer is often portrayed as the processing of information. During this process, software developers use a variety of distinct applications to fulfill their tasks. Often, these applications are used together, especially when transferring information from input to output. Additionally, a software developers' work has been observed as highly fragmented and full of switches between different activities [1, 2]. In the desktop environment, this situation leads to a complex arrangement of ever changing windows. Furthermore developers usually seem to open many more windows than they close which leads to a growing list of open windows [3]. Considering our limited cognitive ability, it seems likely that the complexity of modern desktop environments negatively impacts developer focus and productivity.

Other research shows that context switches, such as windows switches between Integrated Development Environments (IDEs) and other applications, impacts the flow, focus and productivity of a developer [2].

Similar problems arise when too many windows are open since this reduces developers' navigational efficiency as they spend more time looking for the window of interest [3]. It has been know for a while that crowded workspaces pose problems for developers. Roethlisberger *et al.* named the problem the *window plague* [3] and multiple researchers have proposed solutions to mitigate the problem but only within the IDE [3, 4].

In this thesis, we argue that there is no reason for the window plague to be restricted to IDEs, and instead, that it is a phenomenon inherent in every window-based desktop environment. As a window based operating system, Microsoft Windows could be fairly prone to an overly crowded desktop environment with many open windows and tabs. Additionally, we show that by considering the window plague problem holistically at the desktop-level there is potential to further improve the focus of knowledge workers.

1.1 Motivation

To gain insights into complex desktop interaction problems such as the window plague, we need to understand software developers' desktop environments.

A substantial amount of research has been undertaken to understand what software developers do at work [1, 5, 6, 7, 8] and how they use different specific aspects of the desktop environment. Interactions within the IDE, in particular, are well documented [3, 4, 9, 10, 11, 12]. From this research, we know developers spend a significant amount of time interacting with their desktop environment; in the IDE alone, they spend around 14% of their time rearranging, re-sizing or dragging windows [11]. We also know developers spend up to 39.8% of their time outside the IDE [12].

However, we were not able to find published research that focuses on high-level interactions of software developers with their modern window-based desktop environments. Tools with the capability to capture desktop events were introduced at least 18 years ago [13] but they seem to be either out of date, have never been used in a real work environment [14] or have set their focus differently [1]. Recently, Nick Bradley and Felix Grund conducted a small-scale study in which they captured the desktop interactions of four computer science students and conducted a preliminary analysis [15].

In this thesis, we aim to fill this knowledge gap by providing a characterization of professional software developers' interactions with their desktop environment. Furthermore, we investigate the impact of the window plague on developers' focus at the desktop-level, extending the previous work that only considered IDEs.

1.2 Research Questions

We examine the following research questions in this thesis:
RQ 1: *What are the characteristics of a software developers desktop environment?*
RQ 2: *What are the differences between traditional computer interaction data and visual focus?*
RQ 3: *Is there an existing window plague in developers desktop environments?*

To answer these research questions, we developed a tool to capture desktop environment interaction and visual focus. Using this tool, we performed a field study with 12 participants from three different software companies over a combined period of 195 days and conducted semi-structured interviews. We analyzed the data and present the results in this paper.

We found half of the desktop environments had 10 or more windows open while mostly only two were fully visible. The number of open windows grows over the day and we learned most developers do not proactively close windows. We found indications of an existing window plague outside the IDE, although it does not affect every desktop environment.

1.3 Contributions

This thesis makes three main contributions.

1. A tool to capture desktop environment interactions as well as the visual focus of developers.
2. A reproducible methodology for structuring, processing, and analyzing the collected data.
3. The processed data and the results of our study with 12 professional developers.

1.4 Outline

This thesis is divided into eight chapters. Chapter 1 motivates the work and introduces the research questions. An overview of related work is given in Chapter 2. The methodology is formulated, and the participants and interview structure are described in Chapter 3. Chapter 4 describes the monitoring application. Results of the analysis are presented in Chapter 5 and discussed in Chapter 6. Ideas to improve and extend this research are given in Chapter 7. Chapter 8 concludes with a summary of the most important aspects of this work.

Chapter 2

Related Work

Developers have been observed numerous times throughout the years in research with different areas of interest. In this chapter, the relevant related work is laid out. The related work falls roughly into five categories: We will start out with what we know about software developers from an observational perspective and then to go into what is known from observing various participants with monitoring applications. We will then go into the research performed with monitoring applications that observe the IDE of a developer and find out what the *window plague* problem is. After that, we give a short overview of what has been exposed through equipping developers with eye trackers. Lastly, we look at related research that observed parts of the desktop environment of software developers with monitoring applications.

2.1 Manual Observation of Developers

Decades ago, researchers started to conduct empirical studies with software developers. The daily activities of software engineers with their respective frequencies have already been explored in multiple studies a long time ago. A study by Singer *et al.* from 1997 found the most performed activities of a software developer to be those of *reading documentation*, *looking at source code*, *writing documentation or code* and *attending meetings* [5]. The participants were observed switching between these activities. Overall, they found that software developers spend most of their time searching for information and reading documentation.

Already in early 2004, Mark *et al.* and Gonzalez *et al.* have studied task switches, multitasking and interruptions in the work place and continued to do so [6, 7, 8]. Over 7 months they observed and interviewed developers and found they only spend an average of around 3 minutes on a single task before they switch, and an average of 12 minutes in a working sphere (a high-level unit of work or activity) [6]. They also found 57% of all tasks to be interrupted and often fragmented [7]. Soon after, Mark *et al.* found that interruptions and work fragmentation result in negative effects for knowledge workers [8]. Their data indicates people seem to compensate interruptions by working faster and therefore experience more stress and frustration.

A more recent study by Meyer *et al.* investigated software developers' perceptions of software development productivity [16]. They conducted an observation of 11 professional software developers and found that a developer switches tasks 13.3 times per hour on average. Furthermore, they found developers perceive their days as productive when they complete tasks without significant context switches [16].

Since manual observation is time consuming and does not scale well, researchers quickly started to adapt applications that collect the necessary data for them.

2.2 Monitoring the Desktop Environment

Employing monitoring applications that automatically observe participants is not new. Various researchers proposed a large number of applications to capture computer interaction data. Already in 2000, David Hilbert and David F. Redmiles described and classified existing monitoring applications and systems that were designed to track user interface events or individual applications [13]. The documented systems include applications with capabilities for capturing key presses, mouse clicks, mouse drags and other events. Some of the described applications could have provided insight into the use of the desktop environment. Although the basic interaction methods with a window may not have changed much since the year 2000, the desktop environment has undergone many changes due to new tooling and interaction designs. Other examples from 2002 include research by Fenstermacher *et al.*, where they proposed a framework to monitor users in a cross-application environment [17]. However, the proposed framework is aimed at human-computer interaction with the applications and not the desktop environment as such.

In 2005, Dragunov *et al.* from Oregon State University proposed *TaskTracer*, a monitoring application designed to support multitasking knowledge workers in locating and reusing past processes [14]. The application has the capability to collect data about the desktop environment and especially applications like Microsoft Office, Visual Studio and Internet Explorer via an addin. It monitors the desktop environment by collecting events like window focus, clipboard and file creation events. Detailed records of user activity are also collected. Although the application's potential to observe the desktop environment seems to be quite extensive, the author states the application was not tested in actual work environments. The application and recorded data has since been used in multiple subsequent studies, mostly to train machine learning based predictors [18, 19, 20]. Examples include folder and task prediction. However, it does not seem the application or data were ever used to report on the desktop environment use; neither of students, nor of real-world participants.

Around the same time, Hutchings *et al.* performed a study with the intent of comparing single monitor and multiple monitor usage [21]. They deployed a monitoring application to track the window management events of 39 participants and gained some insights into the desktop environment in general. The analysis of the collected data showed that 78.1% of the time participants had eight or more windows open. The average amount of time that any window was active was 20.9 seconds, and from this they conclude that the participants frequently shifted their attention among several windows. Furthermore, they found single-monitor users have an average of 3.5 visible windows while multi-monitor users averaged between 4.1 and 6.8 visible windows. The median was 3 visible windows for single-monitor users and 4–6 visible windows for multiple-monitor users, depending on the size and resolution. Another finding is that, with more screen space, users seem to leave part of the screen empty. Although the research by Hutchings *et al.* shows deeper insights into desktop environment usage, it is not tailored to software developers and might not be as applicable nowadays, 14 years later.

Newer studies often monitor specific applications, tasks or activities rather than the desktop environment in general.

2.3 Monitoring Developers IDE

As the IDE is one of the more significant applications inside the desktop environment of a software developer, many researchers focus explicitly on IDE usage.

In 2005, Mik Kersten and Gail C. Murphy proposed an Eclipse IDE[1] plugin *Mylar* that, among other things, monitors a programmers' activities and captures the relevance of code elements in regards to their task [22]. Through the data of a developer's interactions, the plugin is able to build a task context. In a subsequent study, Murphy *et al.* analyzed the data Mylar provided to give insights into what Java software developers do while using the Eclipse IDE [9]. They collected data from 41 Java developers and reported on the most commonly used navigation patterns, commands, and views.

A recent study from 2015 by Sanchez *et al.* analyzed the data provided by the same Eclipse IDE plugin, that was renamed to *Mylyn* in the mean time [2]. Dozens of developers provided several thousand interaction traces for the study, which aimed at gaining insights into work fragmentation. Corroborating other research, they found that the work of software developers is highly fragmented and full of switches between different activities. Further, they observed that work fragmentation is correlated to lower observed productivity. Specifically, longer activity switches seem to strengthen the correlation. Frequently switching windows may therefore impact focus, flow, and productivity of developers.

Minelli *et al.* investigated on how developers spend their time using an interaction profiler installed into the PHARO[2] IDE, which provides fine-grained IDE interaction data [11]. They observed about 740 development sessions by 18 developers and a total development time of 200 hours. Due to the collection method, the work is mainly focused on the IDE. They found developers spend around 14% of their time rearranging, re-sizing or dragging IDE windows. They furthermore state that the time spent outside the IDE only accounts for about 8% of the time. Additionally, they found the number of switches from inside the IDE to (a window) outside linearly correlated with the code understanding time as well as time spent rearranging windows; therefore having a negative impact on focus.

Ammann *et al.* instrumented the Visual Studio[3] IDE and monitored the interactions of professional developers for more than 6300 hours [12]. They mainly focused on how developers spend their time inside the IDE and the differences to other IDEs. They have a large number of findings but mostly out of the context of this work. One noteworthy result is that developer spend 39.8% of their time outside the IDE, potentially using external tools for their work.

2.4 The Window Plague in IDEs

Roethlisberger *et al.* conducted several empirical surveys and studies with software developers and collected data about their window usage inside the IDE [3]. They found an average number of 16.68 open windows in the Eclipse IDE and 14.29 in the Squeak IDE. Moreover, they discovered that developers open many more windows than they close and therefore let the list of windows grow steadily. Developers mentioned in an interview that their strategy is to let number of windows grow until they are completely done with the current task. After the task, they take some time to manually close all or most opened windows. Only a few developers close deprecated windows regularly during a task. Moreover, developers are aware of the deprecated windows, but they are not willing to manually close them.

[1]https://www.eclipse.org/ide/
[2]http://pharo.org/
[3]https://visualstudio.microsoft.com/

As a result of this, they found developers often lose time or even overview in the crowded workspace, and learned from discussions with the developers that a cluttered workspace with many open windows decreases development efficiency. Roethlisberger *et al.* named the problem of an overly crowded workspace with many open windows within the IDE the *window plague*.

In search of a solution to the window plague, they proposed *AutumnLeaves*, a tool that grays out or closes windows unlikely to be used again. To detect what windows are likely not to be used anymore, the tool assigns weights to open windows and indirectly models references between windows. The weights are adjusted upon user actions using a specially developed scheme. In the the evaluation of the tool, they learned that the tool was able to minimize the average number of open windows. Another important takeaway is the finding that developers are not willing to accept a fully automatic closing mechanism and seem to desire a veto before the windows are closed.

Minelli *et al.* used a tool to mine fine-grained UI events inside the IDE and processed them to be presented visually [10]. During their observation they found evidence of the window plague: They observed a developer cleaning up the desktop environment at regular intervals of about 1.5 hours. In this cleanup phase, the developers closed almost all windows to refresh the environment. The presented visualization shows the growing number of windows until the entropy level of the environment becomes unbearable. The developer is then observed grooming the desktop. The paper notes a task may at that point have been completed or the environment may just have become unbearably convoluted, so that the developer needed to start over again.

Minelli *et al.* further tried to mitigate the observed window plague problem by exploiting the collected IDE interaction data [4]. They propose a novel tool called the *Plague Doctor* that detects the likelihood that an IDE window will be reused in the future and automatically closes low-probability windows. Featurewise, *Plague Doctor* is quite similar to *AutumnLeaves* but has some advantages, most notably, an updated window weighting strategy. The tool also operates on more interaction data than the previously proposed approach.

In contrast, this thesis aims to investigate whether the window plague problem exists outside the IDE. If so, we plan to build the basis for future solutions against a holistic window plague outside the IDE to build on.

2.5 Tracking Developers Eyes

Researchers have also discovered the potential of eye tracking in the observation of software developers and especially in monitoring applications.

Already in 1990, Crosby *et al.* designed an experiment to determine if programming experience influences code comprehension using eye trackers[23]. They found novice and experienced developers display different patterns when reading an algorithm. Later, multiple other researchers were able to gather insights into the differences between novice and expert developers during code comprehension tasks by using eye trackers as well (e.g. [24]).

Other researchers have used eye tracking technologies to look into developers' comprehension of different software artifacts [25, 26, 27, 28]. Some researchers have also evaluated the potential of eye-tracking for detection of software traceability links [29].

Fritz *et al.* conducted a study with 15 professional programmers where they tested whether eye-trackers and other psycho-physiological sensors could be used to measure task difficulty [30]. They found one can predict task difficulty with reasonable precision even for new participants, where using an eye-tracker alone has provided the best predictive power.

Kevic *et al.* gathered interaction and gaze data from 22 software developers working on change tasks [31]. They automatically linked eye gaze data to the underlying source code elements in the Eclipse IDE and found, among other things, that the eye-tracking data captures substantially more, and different aspects of developers interactions.

Through the collection of eye tracking data, we try to gain insights into those different aspects in the use of the desktop environment and try to give an understanding on how they differ from traditional interaction data. It seems, there is only a limited amount of research using monitoring applications outside the IDE to look into the desktop environment use or even the more general computer use of software developers.

2.6 Monitoring Developers Desktop Environment

Recently, Meyer *et al.* performed a study where they installed a monitoring application on the computers of 20 professional software developers from four different companies for an average of 11 work days [1]. The application was used to analyze the types and frequencies of application switches and to correlate developers' work habits with perceived productivity. One of the results is that developers switch activities after 0.3 to 2 minutes minutes on average and therefore interact with their desktop in a highly fragmented way. Furthermore, they note that in rare occasions, developers spend long hours without switching activities. They found the most often occurring activity pattern is to switch during coding tasks to emails. Switching from a coding task to work related web browsing was observed often as well. Additionally they made many productivity related findings, such as that developers can be grouped into morning, low-at-lunch and afternoon people. The monitoring application developed by Meyer *et al.* captures relatively few desktop environment properties compared to the current study. Additionally the current study uses eye tracking features and is more focused towards the the desktop environment data analysis.

In 2017, Nick Bradley and Felix Grund looked into the desktop interactions of knowledge workers as well as possible distractions in the desktop environment [15]. They developed a background application with eye tracking capabilities that was employed to observe four graduate computer science students at their desktop for up to two weeks. They found participants to have a median number of 12 open windows and switch frequently between them with an average of of 326 switches per hour. They found that out of all window switches that took place in under 10 seconds, 40% were even below one second. Furthermore they found that the active window is maximized 67% of the time, indicating that other windows on the same monitor do not pose a distraction. They found the active window to be on the primary monitor 87.18% of the time. Using eye tracking, they found users switch applications much more frequently with their eyes than with their mouse and keyboard and most notably that participants had only looked at the active window for 18.75% of the time. Further results are related to the fixation time of a window. Participants either look at a windows for less than 100 ms or more than one second. Overall, most gazes lasted between 100 and 200 milliseconds.

Although the study provides valuable insight, it has some room for improvement: The authors mentioned it was hard to link the data from the eye tracker and the desktop environment and that mistakes in the linking process may have happened within the tool as well as the analysis. They recognize the need for a better technique in the linking of the data. The desktop environment data was collected with a polling approach, where every 500ms a desktop screenshot was taken. The authors would like to move away from the polling-based approach since it may be prone to miss important environment changes, in particular what events triggered the changes. Furthermore, they would be looking into an event-based data collection mechanism. User presence was not detected and therefore there was no notion of idle time. Furthermore, the authors would like to to categorize applications into sets in the future. Lastly they acknowledge the sample size of four participants is very limiting. The authors consider the study a pilot for a larger future field study, and the paper was never published. It was clear from discussions with the authors that they do not intend to pursue this further.

This thesis is similar in many aspects to the study performed by Bradley and Grund. We try to capture almost the same variables when observing the window environment and analyze similar questions. We aim to extend the study by overcoming the indicated shortcomings in the following manner: The eye tracker data and the desktop environment is linked automatically at run-time by the application to minimize the chance of mistakes. The polling approach is discarded in favor of an event-based approach where all events that cause changes are captured. User presence data is captured in multiple ways and therefore there are multiple notions of idle, such as user presence detected by the eye tracker or mouse and keyboard idle. The analysis categorizes the applications into activities and a big part of the analysis is based on these activities. We investigate a broader participant population where the size is increased from four to twelve and extend the observation period for all participants. Last, our monitoring software is deployed to professional software developers.

Chapter 3

Methodology

To answer the given research questions, an approach to collect the necessary data needed to be developed. We chose two distinct means of data collection. First, we implemented an application that allows us to monitor the user's desktop environment and therefore capture data about their window interactions as well as eye tracking data. Second, we conducted a semi-structured interview with all participants to enrich the collected data with explanations and intent. In this chapter, we will first describe the participants, go shortly into the interview questions, and discuss our means of analysis.

Participants For our study, we were looking for participants who were professional software developers willing to share data and insights into their interactions with the desktop environment. We have restricted ourselves to developers using the Microsoft Windows operating system for simplicity. To find these participants, we leveraged personal contacts as well as the contacts of the Software Evolution and Architecture Lab[1] of the University of Zurich. We found three software companies of varying size based in Switzerland willing to take part in our study. Participation was completely voluntary for the employees. To incentivize participants, we brought small presents in the form of chocolate or confections whenever we visited the companies.

We found 12 professional software developers who were willing to participate in our observational study. All participants were using Microsoft Windows as their operating system and installed our background application that allowed us to observe their desktop environment during the daily work. Additionally, we installed a Tobii Eye Tracker 4C[2] on the primary screen of all participants to capture visual focus. The participants unanimously described development as their primary work area with some having additional secondary work areas, such as project management, system engineering, and test. The role of 9 participants (75%) was best described as individual contributor with some having an additional role as architect. The role of the remaining 3 participants (25%) role was lead with additional roles such as manager, executive or architect. The participants had an average of 17.5 (± 10.6) years of experience with software development, ranging from 2 to 35 years. Out of these, an average of 14.25 (± 9.1) years were professional experience, ranging from 1 to 25.

[1]https://www.ifi.uzh.ch/en/seal.html
[2]https://tobiigaming.com/product/tobii-eye-tracker-4c/

Interviews To gain insights into the participant's use of the window environment and to find explanations for the collected data, we conducted a semi-structured interview with all 12 participants. The interview consisted of 23 questions, which can be found in the Appendix. All of the questions were voluntary and a participant was allowed to drop out at any time. The questions were on the following subjects: 4 on the participants background and experience, 8 on their desktop environment and use, 3 on finding a tab or window and switching to it, 2 about distractions in the desktop environment, 3 on shortcomings and possible improvements of the window environment, and 2 on the study and study situation. Out of the 23 questions, 9 had a closed set of answers, while 14 of the questions were open-ended. Almost every participant answered all questions. All interviews were audio recorded and were later transcribed into a Microsoft Excel[3] table. Further analyses, such as computing the average number of experience years, were performed directly inside the table. Some answers were filtered during the phase of the transcription to remove all personally identifiable information.

Analysis The application was installed for a time span of one to four weeks with 12 individual software developers. This resulted in a total of 195 observed days. The participants uniformly described the observed work weeks as rather usual and nothing out of the ordinary. All participants stated the application did not alter their daily work. Many participants mentioned that they forgot about the application a short period of time after installation. A total of 210,573 desktop environment snapshots with a total of 5,162,321 windows (929,309 unique) were collected. Furthermore 3,987,896 mouse positions and 32,559,307 fixations were collected through the eye tracking feature of the application.

The data was collected from the individual participants' computers and merged into a central database for the analysis. In the process of this merge, every entry was assigned a participant number (p 1 - p 12) based on the originating database. The analysis was conducted in a largely exploratory and descriptive way using R Studio[4]. Due to the heterogeneous work environments of the participants and the large number of used applications, we had to group applications into activities to gain useful insights. A component developed by André Meyer of the University of Zurich was used to perform this grouping. The component was extracted from his self-monitoring tool Personal Analytics[5].

We investigated what indicators of the window plague other researchers identified in their data e.g. [3, 10, 4]. We then tried to recognize the mentioned indicators in our dataset. Through the processing pipeline of the R Notebooks, the research is reproducible. The major findings from the R Notebooks are presented in this paper under Chapter *Results 5*.

[3]https://products.office.com/en/excel
[4]https://www.rstudio.com/
[5]https://github.com/sealuzh/PersonalAnalytics

Chapter 4

Monitoring Application

To gather desktop environment insights from participants, an application to capture the data had to be developed. As a window-based desktop environment, we chose the Microsoft Windows platform. In this chapter we first discuss the particularities of the Microsoft Windows environment. The monitoring application is then introduced and the captured variables are presented.

4.1 The Windows desktop environment

Since the Microsoft Windows desktop environment is common on personal computers, many people are familiar with it. We will discuss the basics shortly to ensure the associated terminology is well understood, since it will be used throughout the paper.

The core of the Microsoft Windows desktop environment is built on the notion of the *window*, a graphical interface to interact with an application. In the following, we will call applications with at least one window *interactive* applications. Numerous interactive applications may be run simultaneously and every application may open up multiple windows. Whenever multiple windows exist simultaneously, they may overlap and a window may partially or fully occlude others. This occlusion is governed by the *Z-order*, which defines the stacking order of windows. Windows higher in Z-order may obstruct lower windows from being seen by the user. At any given point in time, exactly one window is on top of the Z-order. This window is called the *active window* and receives the keyboard input. An open window may be invisible to the user for another reason than being obstructed by others: The user has the ability to *minimize* a window, or in other terms, hide it from the desktop and reduce it to the taskbar. The *taskbar* is a special window that is located along one edge of a screen and provides an overview of the existing windows in the environment. Also, the taskbar provides a special section to display running background applications called the *system tray*.

The *window manager*, is a main component in the desktop environment, which controls the aspects and attributes of windows. One attribute that we have already talked about is its Z-order. Some other important attributes associated with a window are the following: The *title* or interchangeably called window name are used for identification (in conjunction with other attributes, such as the handle, style or window class). The *position* and *size* of a window determine its placement on the desktop. The *show state* of a window determines whether it is minimized, maximized or always on top of all other windows. Finally, the *application* where the window originates is another important attribute with its own associated properties. There are many more attributes associated with a window like relationships, menus etc., that we will not go further into.

The window manager also allows the user to interact with a window. There are six primary interaction methods with a window that determine its life cycle:

- *Create* is performed when a window is first opened. It shows up on the desktop.
- *Activate* moves a window into the foreground and on top of the Z-order. It becomes the active window. A term we use interchangeably to activated is *being switched to*.
- *Minimize* (sometimes called iconify) reduces a window to the taskbar and makes it invisible to the user.
- *Restore* returns a reduced window to its previous position and makes it visible again.
- *Move or Resize* (includes maximizing) changes position or size of a window.
- *Destory* closes a window.

There are other interaction methods with a window, e.g. *Rename* that changes the name (or title) of a window. These methods are usually not performed by the user himself, but rather by an application and are therefore not relevant for this thesis. There are other particularities of a desktop environment, such as logical workspaces such as virtual desktops. We will not look into these. Also, we will regard invisible windows that are not occluded or minimized as closed for simplicity. Examples of such invisible windows are cloaked Windows 10 Store apps or message-only windows.

4.2 Implementation

As discussed in the Methodology (3) chapter, we did not know of an application that could provide us with all the information we were interested in. It became apparent that we had to build one for this specific task. As a window based desktop environment to observe for our study, we had chosen the Microsoft Windows platform. With this prerequisite, we decided to develop our monitoring application in the C# programming language that allows us to use native methods of the Microsoft Windows API (window manager) and still abstracts low-level details. The application should always be running in the background and not require user interaction. We therefore originally implemented the monitoring application as a stand-alone background application that was only visible as a system tray icon. In a second endeavor, the application was integrated into *Personal Analytics*[1], an existing self-monitoring application developed by André Meyer at the University of Zurich. Personal Analytics also resides in the system tray and does not require any user interaction. Furthermore, we prevented Personal Analytics from showing any windows for the duration of the data collection. This was done to ensure the process of data collection would not influence the behavior of the participants and ultimately does not distort or influence the collected data.

Further requirements for our monitoring application resulted from the related work (see chapter 2). As the authors of existing monitoring applications had recognized minor shortcomings to their applications [15], we had to ensure we would not retain those. Some of these requirements were as follows: The eye tracker data and the desktop environment had to be linked automatically at run-time by the application. This ensured data was matched correctly. We had to implement an approach where all user interactions with the desktop environment would cause events that could be captured and named. User presence data had to be captured and we chose to implement multiple possible notions of user presence. We implemented user detection by the eye tracker or mouse and keyboard interaction detection.

[1]https://github.com/sealuzh/PersonalAnalytics

4.3 Captured Data

The application has the capability to capture a variety of data around the desktop environment and interactions as well as fixation data. The data is stored into a local SQLite[2] database on the participants computer and has to be manually collected from there. Only after the collection are the individual databases combined into a central database for later analysis. To combine the individual databases, a special tool was developed that kept the relations in the data intact and assigned participant numbers to the obtained data. In the following, we will lay out the most important data that was collected.

Desktop Environment Data Since our main interest is directed at the desktop environment, the most important data captured by the application are the *window* attributes. The application tracks almost every property of a window, including title, class, position, size, status, and hierarchical information. Along with the window, the associated application is captured as well as most associated process information. To further support our window data, we gathered additional information with relation to the window, such as monitor data. The sum of all collected window information at a given point-in-time form a snapshot of the desktop or *desktop snapshot*.

Traditional Interaction Data To gain a more dynamic understanding of the desktop environment interactions than provided by static desktop snapshots, the application observes the window manager interactions. All primary interaction methods (*Create*, *Activate*, *Minimize*, *Restore*, *Move or Resize*, and *Destory*) as well as some indirect ones (such as *Rename* etc.) require a new snapshot be created. The developed application captures all these window manager interactions and creates new desktop snapshots after every interaction. Additionally, the application captures *Mouse Position* as further interaction data, as well as a measure for when the user was last active at the computer. Moreover, we collected data about other specific events, such as when data got copied into the clipboard, when the monitors or settings were changed.

Visual focus To capture the visual focus of the users, the application is capable of eye tracking. Usually eye trackers provide the screen coordinates of the position where a user is looking at in regular intervals. The interval duration depends on the eye tracking hardware. Every individual data point collected in this manor is called a *Gaze* data point and is mostly an artifact of the employed eye tracking hardware. We were interested in the visual attention of the user and therefore in positions where the user maintains the visual gaze on a single location. This is captured in so called *Fixation* data. The movement in between two individual fixations is called a *Saccade*. We used the Tobii Core[3] Software development kit (SDK) to obtain mainly fixation data.

As we were targeting the collection of desktop environment data, we used the provided screen coordinates of the eye tracker to derive the window a user was looking at. The application captures the fixated *position* as well as a link to the associated *window*. Further data obtained from the eye tracker includes *eye position*, *head pose* and *user presence*.

A major shortcoming to the used eye tracker and associated SDK is the fact that only the data from a single monitor may be captured. We decided to deploy the eye tracker on the primary screen of the participants. As a result, we were only able to capture visual attention on the primary screen.

[2]https://www.sqlite.org/

[3]https://developer.tobii.com/

Chapter 5

Results

The developed application helps to address the research questions as it gives insights into the characteristics of a desktop environment, the associated interactions and focus.

In this section, we start off with a description of the characteristics of a developer's desktop environment. We then present the interaction patterns that developers exhibit when working with the desktop environment. Following that, we present data on how developers direct focus throughout the daily work, how it compares to traditional input, and what might have a negative impact on said focus.

5.1 A Developers Desktop Environment

To better understand developers' usage of their window environment and how we might be able to support focus at work, we first examined the participants desktop environments.

In general software developers use 9 applications. When developers first log in into the desktop environment, there are usually no applications running and they start out by opening the applications necessary for work. By the time all applications are started, there are a median of 9 ($\pm IQR$ 4.0) interactive applications (with a visible representation) open simultaneously. We looked at how many distinct applications an individual developer uses over time. The lowest number of distinct used applications observed during time of our study was 17, while the participant with the most distinct applications used 115 (median 35 $\pm IQR$ 12.5). Overall we observed 291 distinct interactive applications that developers use at the workplace. The maximum number of simultaneously opened distinct applications that we could observe was 19. We found our participants spend their time mostly developing (38.9%), browsing (20.7%), emailing (9.6%) and working with documents (9.6%). All applications are presented with a visual representation for the developer to interact with. In a window based desktop environment such as Microsoft Windows, the visual representation of the application is the window.

Developers have a median of 10 open windows. The median 9 applications create a median of 10 ($\pm IQR$ 10.0) open windows on the developers desktop at all times. This result fits other research, where participants had 8 or more windows open for most of the time [21] or where a median number of 12 ($\pm$0.4) open windows was observed [15]. The distribution of open windows is skewed right with a fair amount of outliers and therefore the higher average number of 15.2 ($\pm SD$ 14.6) seems less meaningful. We captured a maximum number of 95 simultaneously open windows.

The collected data shows that every application on a developers desktop opened around 1.11 windows on average. The applications that opened most windows were development environments, console applications and file explorers. Every window is 993 ($\pm IQR$ 1411) pixels wide and 519 ($\pm IQR$ 882) pixels tall in the median.

Usually, a median of 2 ($\pm IQR$ 3.0) windows are maximized. From another perspective: Out of all observed windows, 14.4% were maximized. Active windows seem to be significantly more likely to be maximized than others. Out of all observed active windows, 46.7% were maximized. Other research has shown that an active window may even be maximized for up to 67.12% of the time[15]. Additional to the median of 10 open windows on a desktop, there is a median of 2 ($\pm IQR$ 3.0) windows minimized. This means developers permanently retain two additional windows that are hidden from the desktop environment.

We were interested how the number of open windows differs among different developers.

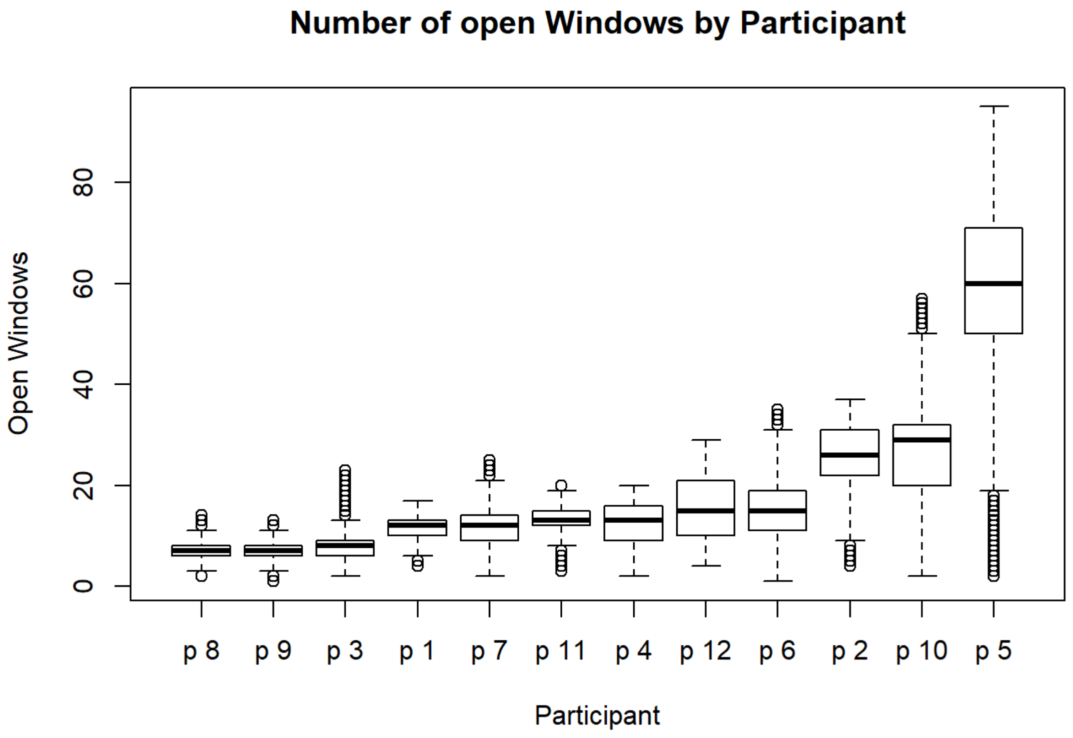

Figure 5.1: Boxplot of open windows by participant ordered by median

The number of open windows differs significantly between developers We observed the number of simultaneously open windows to vary significantly from developer to developer. Figure 5.1 shows the distribution of open windows over all participants in the form of a box plot ordered by median. The box plot was calculated over all captured desktop snapshots. The median per participant reaches from 7 to 60. There are a median (of medians) of 13 ($\pm IQR$ 6.7) open windows across all participants. This median of medians is slightly higher than the median over all windows due to the relatively high numbers of windows a few participants used. The boxplot clearly shows how participant 5 usually has significantly more open windows than most other participants. We were interested as to why this single participant's window count was of the observed magnitude and investigated the desktop environment in detail. We found participant 5 to be using different tooling than most other participants. The different tooling included many console windows that often piled up on the desktop in numbers that were never observed in other participants.

We then looked into how the median number of open windows per developer evolves over different days and found it only marginally fluctuates for most participants. The average standard deviation for all participants between different days was 5.7 windows. When excluding the two participants that had high fluctuations, we observe an average standard deviation of 1.5 windows between different days. We were not able to find a correlation between the median number of windows and the seniority of a developer. In general, the significant difference between the participants may be attributed to a variety of circumstances. We found that the applications used, and therefore to some extent the current task, influences the number. Furthermore we found participants to use contrasting desktop environment clean up strategies that might lead to a difference in number of open windows. The observed cleanup strategies are further discussed in Section 5.2.

We looked at how the number median of windows evolves over the work day.

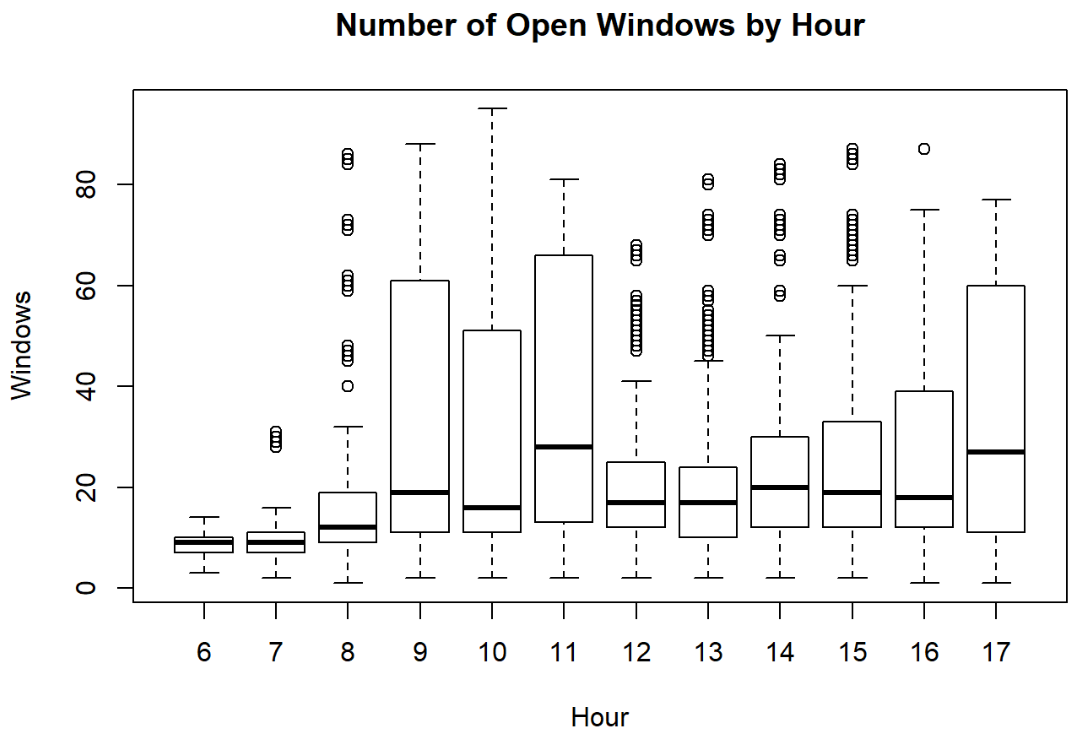

Figure 5.2: Boxplot of open windows by hour

The number of open windows grows slightly over the day. Over the course of the day, we noticed a slight increase in the number of open windows. For this analysis, we focused on the time where we had data from all participants, namely from 8 am to 5 pm. Figure 5.2 shows the distribution of open windows over the course of the day during these work hours. The figure shows how the median number of windows increases slightly until lunchtime. By that time, the number of windows decreases and starts to rise again in the afternoon. We observed a median of 12 ($\pm IQR$ 53) windows open at 8 in the morning with a maximum of 86 opened windows. In the afternoon at 5 pm on the other hand, we detected a median of 27 ($\pm IQR$ 22) open windows with a maximum of 77 open windows. We analyzed the window growth per participant and day in the time span and found an average window growth of 4 ($\pm SD$ 8.8) windows in the time span. While we observed the growth of windows was manageable with only a few windows open, it naturally had a bigger effect with more open windows and resulted in the biggest window growth of 38 windows.

The observation of growing numbers of windows over the day matches the observations from within the IDE made by other researchers [3]. The growth over time may at some point result in more open windows than one is cognitive capability may handle.

This mainly happened over the course of the work hours, but what happens over night?

Many software developers do never turn off their computers. Since opening up applications and ordering windows is a time consuming process, most developers prefer to let the computer run or use hibernation whenever they leave the computer. As a result they never have to re-open all applications when they come back to the computer. Many developers let their computer run overnight and over the weekend and will only restart them when absolutely necessary, such as for updates or to fix problems. This leads to situations, where the number of windows grows over multiple days. This behavior was not consistent and the data does not allow us to determine whether an update or problem was experienced, but overall we observed 7 of the 12 participants using this technique regularly. In the interviews, multiple developers stated that this technique is used to eliminate the setup time for their applications and window arrangement when they return to the computer. Furthermore, it helps them to relate to the task or activity performed before they left. We observed a minority of developers who shut down their computers in the evening and re-opened their applications every morning. The behavior may be related to company policy.

Most windows are not visible. By intersecting the rectangle that a window spans with the other opened windows along the Z-Axis, we were able to determine which windows were visible. From that, we calculated the number of fully visible windows on a desktop. We performed this operation for all captured desktop screenshots. Windows that are partially obscured by overlaying windows are therefore not counted as fully visible. The resulting data shows there are only a median of 2 ($\pm IQR$ 1.0) windows fully visible. The most concurrently fully visible windows we observed were 6. Therefore, there may be a median of 8 obscured and 2 minimized windows on developers desktops. Since we only looked at fully visible windows, some of the 8 obscured windows might be partially visible.

Developers usually lay out their ever growing list of windows on multiple monitors, which is why we were interested to learn how developers use their monitors and how windows are distributed between these monitors.

77% of all windows are on the primary monitor. At first we looked at how many monitors were used. 10 (83.3%) of the observed developers work with two monitors, the other 2 (16.7%) work with three. We were not able to observe a professional developer with a full-time single-monitor setup. There were some developers utilizing a laptop with external monitors and therefore switching between single- and multiple-monitor setups. This seems to corroborate other research, where three out of four participants utilized two monitors and a single participant used three monitors [15].

One of the monitors is a *primary monitor* and for simplicity, we consider the others to be *secondary monitors* in the following. Microsoft Windows defines the primary monitor to be the one where the left upper corner receives the virtual screen coordinates (0/0). For the developer, the primary monitor is the one that she/he uses the most time and where 77.2% of all windows are opened. Only 22.8% of the windows are being opened on a secondary monitor. Other research shows numbers of the same magnitude and even suggests this could even be more biased towards the primary monitor (87.18%) [15].

For active windows, this number only marginally differs (69.8% on the primary). Maximized windows are distributed almost equally between the monitors (49.25% on the primary). This uniform distribution of maximized windows between monitors seems intuitive, since only one maximized window can be visible on a single monitor at the same time. We have before found, that there is a median of 2 maximized windows on every desktop. When we combine these two findings, we could state: Most times one window is maximized on the primary as well as on the secondary monitor. This would however not directly imply that there are only two visible windows, since other windows can be further up in the Z-Axis and therefore in front of the maximized windows.

Some windows have a fixed position. We observed some applications with windows that seem to have a more or less fixed position in the developers desktop environment. The windows of these applications are almost never moved, minimized or re-sized but only get created and destroyed. With a fixed position, they stay almost always on the same monitor, often maximized.

In the interview, we asked the participants about their strategy of assigning windows to a monitor. Multiple strategies were mentioned. A particular one was mentioned multiple times: There are certain applications with a *fixed position* and size within in the window environment. Then there are *free floating* applications where the window moves around within the window environment and therefore changes position and size regularly. It allows them to align the window of a fixed and a free floating application next to each other. The participants, who mentioned this strategy, explained that this approach is mostly selected for the ability to transfer information from one of the fixed position applications to a free floating one or vice-versa. They argued that with this approach, it is always clear where in the desktop environment a window is and therefore eliminates the need to search for windows. The given explanation fits the results from the data perfectly.

We looked into the data to find out what applications were mostly free floating and which were more of the fixed position type. For this, we generated in a ratio, that represents how much an application was moved compared to other activities. This ratio was calculated by dividing the number of move or re-size events by the total occurrence of an activity. We received an ordering from the most free-floating to the most fixed activities. By far the most often free-floating activities were work-unrelated Browsing and work-related Browsing. Followed by reading and writing documents as well as emailing and planning activities. On the other side of the spectrum were the more fixed position applications, such as debugging and explorer navigation, as well as instant messengers and remote desktop activities.

A fixed position of a window implies that the window was almost exclusively used on a particular monitor (the monitor may vary per participant). Free floating applications are often moved around and were therefore observed on multiple monitors. We binned all programs into a related activity category and observed how many times the category is to be situated on a particular monitor.

Table 5.1: Activity-windows per monitor

Activity	# Total	# Primary	% Primary	% Other
Gaming	642	642	100.0	0.0
Debugging	52071	44799	86.0	14.0
Development	1542702	1242153	80.5	19.5
Work-related Browsing	194761	152995	78.6	21.4
Remote Desktop Use	11063	8492	76.8	23.2
File Explorer Navigation	715135	544116	76.1	23.9
Instant Messaging	268271	199742	74.5	25.5
Version Control	166455	122785	73.8	26.2
Planning	60650	43483	71.7	28.3
Work-unrelated Browsing	25276	17931	70.9	29.1
Emailing	306249	205321	67.0	33.0
Reading/Editing Documents	212431	140557	66.2	33.8
Code Reviewing	398	193	48.5	51.5

Development is almost always done on the primary monitor. Table 5.1 shows the affiliation of an activity with with an individual monitor. The table shows that developers prefer having the IDE, debugger, and work related browser on the primary monitor. The monitor affiliation of the debugger and the IDE imply code is written mostly on the primary monitor. Reviewing seems to be an activity performed in windows that preferably take place on a secondary monitor. Given that windows are generally more likely to be positioned on the primary monitor, activities such as file navigation or collaboration seem to be done more in free floating applications that use both monitors.

Monitors come in different size and resolution and there is a significant difference in monitor size between the different developers. The smallest monitor in terms of resolution we could observe was in WXGA (1366 x 768 pixels), while the biggest one was in a 5K (5160 x 2160 pixels) resolution. The latter (5K) having 10 times (or 1062%) the amount of pixels available on the smallest one. The size of an individual pixel on the bigger monitor was smaller and therefore the display is not 10 times as big.

The sum of the resolution of all available monitors we call screen real estate. As suggested by other research, having more screen real estate may improve manageability of windows, allow for more simultaneously open windows and even allow for more direct access of the windows [21].

Bigger Monitors do not lead to more windows. The idea that somebody with more screen real estate can have more windows open intuitively makes sense. To detect if there is a correlation in our data, we conducted a comparison of the screen real estate in pixels and the median number of open windows. We were not able to find a correlation. At first we looked into the Pearson's product-moment correlation between the median number of open windows and screen estate per developer, which leads to a correlation value of -0.190. Since only a small number of developers participated in the study the degree of freedom is small and we were not able to reject the null hypothesis with a p-value of 0.5. Since some participants also switched monitors during the observation, we then compared the median number of windows per hour per developer with the screen real estate in pixels. This enlarged the set of observations. Using this method we were able to find a Pearson correlation coefficient of -0.268 with a degree of freedom of 1134 and a p-value of $1.3e - 11$, meaning participants with more screen estate generally did not have more windows open.

5.2 Desktop Environment Interactions

After learning about the desktop environment, the next step towards fostering developers focus is to understand how developers use the desktop environment. For every desktop environment interaction, our application captured events and stored them into the database. These events give us a comprehensive understanding on how developers interact with the environment. In this section, we will lay out the data that our application provided as well as further insights obtained from the interviews.

Windows are rarely moved or resized. As discussed in the *Methodology (3)* chapter, there are six primary interaction methods with the window manager and therefore the desktop environment in a broader sense. Our monitoring application captured events for the following primary interaction mehtods: *Create* when a window is first created, *Activate* when a window gets switched to, *Minimize* when a window is reduced to the taskbar, *Restore* when a reduced window returns to its previous size and position, *Move or Resize* (includes Maximizing) where a window changes position or size and lastly *Destory* when a window is closed.

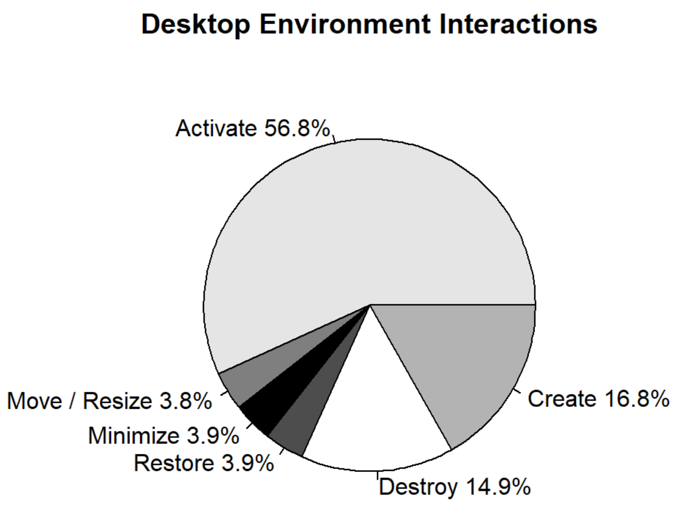

Figure 5.3: Pie diagram of developers desktop environment interactions

A comparison between the frequencies of observed desktop interaction methods gives insight into how relevant the respective methods are for a developers daily work. Figure 5.3 shows a pie diagram of the desktop interaction methods by frequency.

As seen in the diagram, the activation of windows is 56.8% by far the most often performed desktop interaction, meaning we observed many more windows being activated (switching between windows) than any other interactions with the desktop environment. This is followed by the creation (16.8%) and destruction (14.9%) of windows. In comparison, windows are only very rarely minimized (3.9%), restored (3.9%) or moved / re-sized (3.8%).

As one would assume, the frequency with which windows are minimized and restored is about the same. What seems rather unexpected on the other hand is that the number of created windows is slightly higher than the number of destroyed windows. This translates to the fact that there are windows being opened that the developers never manually close. These windows must get closed by other applications, by application crashes or when the computer is shut down. This finding further supports the finding that some developers let the list of windows grow without manually closing them. Similar findings from within the IDE was made by other researchers [3].

When we look further into the events of window creation and destroy, we can determine how long a window is open.

A window stays open for an average of 5.8 hours A developers' applications and their windows may generally stay opened for several hours or days. We have noticed many windows staying open from when they were first launched until the end of the day. Some windows stay open much longer and we could observe windows staying open up to 11 days. Overall, we observed window being open for an average of 5.8 ($\pm SD$ 26) hours. The distribution is skewed right as a result of the high number of windows that get opened up only for a very short period of time.

Overall, windows were active for 59.1% percent of the time that they were open. This high figure is due to the fact that there are many windows open only for a short period of time and active during 100 percent of this time. If we look at windows that were open for over 10 seconds, the active ratio drops down to 32.9%. In general, the longer a window is open, the smaller is its active ratio.

With the activation of windows being the most often performed interaction of a software developer, we were interested to investigate it further. The window activation data can also help us to determine if windows are laying dormant for longer periods of time without being used. We consider two notions of being used: The first one is being active and having the keyboard input associated that is discussed here. The second notion of being used is having visual focus, which we discuss in Section 5.3.

Windows are active for a few seconds at the time. During the time the developers were active, we found that windows are only active for an median of 5.0 ($\pm IQR$ 20.4) seconds at the time. The distribution is heavily skewed right and the average time a window is active is 30.4 ($\pm SD$ 88) seconds. Nonetheless, this low number implies a high number of window switches, especially short-term switches between applications. The longest time a window was active without switching to another window was 38.7 minutes (during the time a developer was active at the computer). Many window switches happen in a very short time period under a few seconds. 21.6% of all windows are activated for a period shorter than 1 second. Out of all windows that were activated for under 10 seconds, 34.8% are activated under 1 seconds. This finding corroborates other research. Bradley at al. note out of all switches under 10 seconds, 40% were under 1 second [15]. The developers activated windows with a median of 39 ($\pm IQR$ 74) window switches per hour. Interestingly, the number of switches per hour seems to be mostly of the same magnitude between the participants. Due to the skewed distribution, this leads to an average of 60.2 ($\pm SD$ 64.3) switches per hour. If we only look at the active hours 9-18, we get a median of 42 ($\pm IQR$ 78) with an average comparable to the one over all hours. We observed a high of 498 window switches per hour. To further illustrate this number, it comes down to a switch every 7.3 seconds. Although the observed number of switches may be regarded as rather high, other research even higher numbers such as an average of 326 ($\pm IQR$ 320) switches per hour were observed [15]. Roethlisberger *et al.* have observed a high number of switches within the IDE and have related it to indications of lost overview and confusion from an overload of windows in the development environment [3].

Developers often switch between the IDE and the Browser The most switches between multiple windows of a single application were observed inside the IDE. Developers seem to rapidly switch between different code windows. Table 5.2 shows the most performed switches between

Table 5.2: The most frequent window switches

Activity A	Activity B	# From	# To	# Total
Development	Work-related Browsing	2176	2229	4405
Development	Reading/Editing Documents	1344	1380	2724
Development	File Explorer Navigation	1143	1133	2276
Development	Emailing	979	968	1947
Development	Instant Messaging	682	675	1357
Development	Version Control	666	652	1318

different applications. We have observed developers switching most often between the IDE and work related browsing. They also frequently switch between the IDE and documents, between the IDE and File Navigation or between the IDE and Emails.

Active windows are mostly alternating. We looked at the active windows over time as a time series of active windows. The series consists of an order of windows that were active in time from the current window backwards. Given this time series of active windows, where x is the currently active window and x-1 the previously active window, x-2 the window that was active before that etc. We looked through our data at how many times these past windows were chosen to be the next window, x+1. Or in other words, we were looking how probable it was for the series of past active windows to be chosen as the next active window. Figure 5.4 shows the results of this analysis.

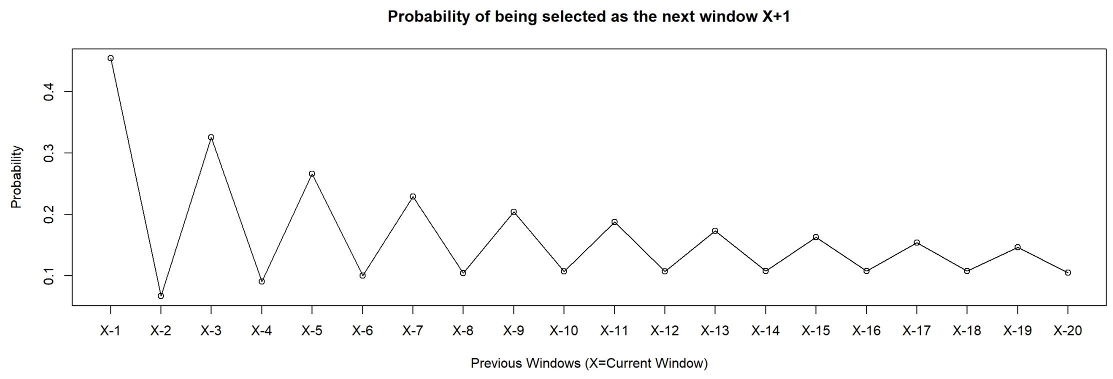

Figure 5.4: Probability of a past active window to be selected as the next active window

The figure shows that the previous window (x-1), has the highest probability to be chosen as the next active window. In 45.4% of all cases, the participants switched to the previous window. In only 6.7% of all cases, the window before that (x-2) was chosen. The window x-3 has a probability of 32.5% to be activated. The windows before that keep alternating in their respective probabilities. In this configuration it is possible that multiples of the last active windows are the same. Our observation shows mostly the oddly indexed windows (x-1, x-3 ,...) and the even indexed windows are the same window. Rarely is the same window indexed as even and odd number. This could indicate that the observed participants mostly switch between two windows and rarely switch back to the window used before that.

More windows do not necessarily lead to more switches. We analyzed the correlation between open windows and number of switches to see if developers with more open windows switch more often. For this we compared the median of opened windows per hour with the number of switches observed per hour. We did this for all the participants. We observed a Pearson's correlation coefficient of 0.03 with a degree of freedom of 1017 and therefore failed to reject the null hypothesis with a p-value of 0.372. We do not have evidence for a real correlation between the total number of windows and the number of switches between them.

At some point, the opened windows need to be closed. We looked into the data and interviews to find out more about the strategy behind closing windows.

Desktops are cleaned up from time to time. We could observe occasions, where many windows were closed in rapid succession of each other. Participants were sometimes closing down the majority of the opened windows in a short period of time.

Minelli *et al.* made a similar but slightly more pronounced observation of the phenomenon inside the IDE and visualized it graphically [10]. They found developers cleaning up their environment from time to time and attributed this phenomenon to the window plague. Furthermore they noted two possible triggers for such a cleaning of the environment: Either a task was completed, or the environment was too cluttered.

This discovery matches other research where interviews found that most developers let the list of windows grow until the task is completely done and only then start to close windows [3].

The data collected by our application does not allow us the insight into development tasks and we did not find any other indication for when desktops were cleaned up in the data. We were looking for another way to get insights into the actions resulting in the observed data and conducted semi-structured interviews with the participants. In the interviews, we incorporated questions regarding this the phenomenon.

Developers almost never close windows proactively. We have asked developers about their strategy on closing windows. Only a small minority of 2 out of 12 developers (16.6%) stated they would close down windows proactively and instantly after they don't use them anymore. 2 participants (16.6%) stated they would clean up the desktop environment whenever they were switching from one task to another. 3 participants (25%) answered, they close down windows when the desktop environment gets too cluttered. The other 5 participants (41%) stated, they would only close down windows for exceptional reasons or when absolutely needed, such as to free memory, for updates or before they leave the computer for a long time.

The developers that close down windows proactively reasoned that the chosen strategy allowed them to maintain overview. One developer even stated explicitly, that he/she closes windows when he/she feels too much time is lost for switching while looking for the next window. On the other hand, the majority of developers that do not to close down windows manually stated that a high number of open windows is not an issue to them.

Developers seem content with their desktop environment. We asked the participants if there is something distracting in their desktop environment. Four participants stated that notifications are a big distraction. Others mentioned various particularities, such as transparent windows and focus grabbing applications. Six participants did not find anything distracting in their windows environments. In the end one participant even mentioned that his/her surroundings, such as colleges and the work environment are far more distracting to him than anything in the desktop environment.

Furthermore, we asked the participants what improvements to their windows environment they would favor. Diverse answers were provided. Some developers mentioned they would like to navigate solely by keyboard and would prefer never having to use the mouse. Others disliked the updating procedures. Twice, participants mentioned a virtual desktop implementation as known from other operating systems but Microsoft has since introduced some improvements concerning virtual desktops.

When asked on how their workflows may be improved, the participant answers varied a lot as well. Answers were ranging from disabling all notifications in the desktop environment to minimizing external factors like meetings. One particularly interesting subject was brought up by two participants: The integration of personal devices, such as the smartphone into their workflows and desktop environments. With the bring your own device (BYOD) policies that some companies implement, the desktop environment is about to be extended onto other devices. A developer mentioned frequently using her/his smartphone as an extension to his/her desktop environment. They use it, for example, to continue working while the computer is busy doing computations. They stated the need for better integration into the desktop environment, such as the possibility to share information.

Overall, only five of the twelve interviewed developers (41.6%) stated that they sometimes or rarely lose time when looking for the next window. The other participants (58.8%) stated this would never happen to them.

For focus it looks similar: Most developers (58.8%) stated in the interview, it never happened that focus is lost when searching for a window. Only five developers (41.6%) stated to having the problem rarely or sometimes. One developer mentioned this could only happen when she or he is not focused anyway.

When speaking about focus, visual attention is an important aspect to examine. The captured fixations give a good indication of visual focus.

Table 5.3: The most frequent fixation switches

Activity A	Activity B	# From	# To	# Switches
Emailing	Work-related Browsing	1225	1134	2359
Development	Work-related Browsing	539	566	1105
Development	File Explorer Navigation	285	325	610
Reading/Editing Documents	Work-related Browsing	286	304	590
Development	Reading/Editing Documents	288	271	559
Work-related Browsing	Work-unrelated Browsing	271	242	513

5.3 Visual Focus

To determine what developers focus visually, we captured eye tracking data and linked it to the desktop environment. We tried to gain deeper insight into how developers direct their focus through the window environment and how traditionally collected environment data differs. In the following, we describe what we learned from the collected eye tracker data.

Due to our collection method, we were only able to capture visual attention on the primary screen. Out of the 32,559,307 fixation points, only 85.6% could be mapped to a window. A window may not have been be mapped if it wasn't recorded as such. Examples of such unmapped fixations are those directed at thumbnails in the taskbar, or in the case of a fixation, the desktop background. Another particularity of the collection method is that there is a small time frame after a window is first created where no eye tracking data can be mapped onto that window until it is fully processed.

Visual Focus is shifted frequently. In the desktop environment, we had observed many window switches and found windows were only active for few seconds in the median. Visual attention shifts seems to be equally as frequent than active window switches. Developers seem to switch the fixated window constantly. Overall, the developers looked at a window for a median of 6.53 ($\pm IQR$ 26.8) seconds. Again this distribution is heavily skewed right. This means the participants were looking at the same window for an average of 35.9 ($\pm SD$ 91.3) seconds. This duration fits in well into the average time that a window was active. The average time a window was active is slightly lower but was computed over all windows on all monitors while eye tracker data is only available for the windows on the primary monitor. The longest time we observed a developer looking at the same window was 42 minutes. Most windows are only looked at for a few seconds before the participant fixates on another window. Windows that were only looked at for under 1 second account for 15% of all fixations.

Developer look at the IDE almost 50% of the time We were investigating what applications developers look at for the most time. Out of all collected fixations, 48.9% were directed onto code in the IDE. 13.8% of the fixations were directed at the browser with work related context and further 8.6% were looking at documents. Another 8% of the fixations pointed at Emails and 6% were directed at the file explorer. The remaining 14.7% of the collected fixations were split between various other activities.

A further point of interest was how developers shift their visual attention. Table 5.3 illustrates differences between what activities developers mostly shift their visual focus and attention. Contrary to the most performed window switches, the most often performed switches in visual attention were observed between emailing and work-related browsing. Only after that, the switch between the IDE and work related browsing was observed, followed by the fixation switch between the IDE and the windows explorer.

Table 5.4: Most frequent fixations away from the active window

Active Window	Fixated Window	# Fixations	% Fixations
Development	Development	918036	3.2
Development	Work-related Browsing	414848	1.4
Development	File Explorer Navigation	324773	1.1
Work-related Browsing	Development	273048	1.0
Emailing	Development	258108	0.9
Development	Instant Messaging	206758	0.9
Total (non-active fixations)		5835115	20.9

The discrepancy to the table of window switches (5.2) may originate from the single monitor limitation of the eye tracker.

79% of all fixations are directed at the active window When capturing traditional computer interaction data, an important measure for the currently performed activity or task of a user is the active window, where keyboard input is received. It is evident that developers do not always look at the active window. With multiple open applications, there is a chance that another, non-active, open window demands visual attention. In pursuit of answering the related research question, we were interested in how fixations relate to the active window.

In our observation, out of all captured (and mapped) fixations, 79.1% were directed at an active window. The other 20.9% were directed at windows that did not have the keyboard input. Participants looked away from the active window at least one time from 67.7% of the active windows. When they did so, they looked at an average of 2.2 ($\pm SD$ 3.2) distinct non-active windows. There were however 32.3% of the active windows where participants never looked away from.

This raises the question of what developers are looking at when not the active window. Table 5.4 shows the most often fixated activities when looking away from the active window. When developers are actively developing, they most often look away to another window with development context, work-related browsing, or the file explorer. Also they often look away from work-related browsing or emails to development applications. Rather often, developers seem to be looking away from development applications to an instant messenger. The table also shows the total number of fixations that were observed away from the active window.

Furthermore, the participants fixated maximized windows about half the time (52.4% of all fixations).

Developers often track the cursor. A further, traditionally collected data point, relates to mouse and cursor interaction data. We were curious to see how cursor data relates to visual focus. At first we looked if the participants were fixating the same window that the cursor is on. We found in 81.6% of the time the fixation and the cursor are directed onto the same window. The average difference between fixation and cursor coordinates is only -88 ($\pm SD$ 570.4) pixels on the X axis and -3.4 ($\pm SD$ 314.2) pixels on the Y axis. The median distance between a fixation and the cursor was only 330 ($\pm IQR$ 471.5) pixels. This means the cursor and the eye were 66% of the time in the same bounding box of 500 pixels. However, does not mean the cursor is necessarily a good predictor for the fixation.

The cursor may indicate the fixation movement. Since the setup of eye tracking solutions is a burden for larger studies, we investigated whether the cursor position may be used as an approximation for visual attention. We probed if the cursor position and the fixation position correlate. For this, we used the input position in X and Y coordinates from the eye tracker as well as from the collected mouse input. We found a Pearson's correlation coefficient of 0.78 for the X-axis and 0.62 for the Y-axis. Both were computed with a degree of freedom of 28080000 and therefore we were able to reject the null hypothesis with a p-value of $2.2e - 16$.

5.4 Threats to Validity

The aim of this paper is to provide a high-level overview on how professional software developers interact with their desktop environment.

Concerning internal validity: While the results may provide a consistent impression of how participants used their desktop environment, there are some shortcomings with the developed monitoring application as well as the performed analysis. The biggest shortcoming of the monitoring application is the fact that the eye tracker only observes a single monitor. As a consequence, we are not able to give complete insights into developers' fixations and therefore separated the eye tracking results from the rest. Another small limitation is the short processing time span that newly created windows have to undergo before the eye tracker data can be linked with them. Due to this and the nature of the employed eye tracking in general, we may have missed a some of the fixations. We found the analysis of the rather big data set was quite complex and resulted in thousands of lines of code. Since the code has never been reviewed, it is likely that some minor mistakes were made during this process. As a mitigation, the code resulting in claims presented in this paper was checked twice. Furthermore, the time span of the observation differed somewhat between the participants. Nonetheless, the data was accumulated across all participants. This may have skewed some of the results slightly.

As for the the external validity: The results of our study may not necessarily be generalizable for the general population of software developers. This is due to the small number of observed participants, the small time frame of data collection, the employed participant selection process (self-selection), and the clustered geographical distribution of the participants. Nevertheless, we tried to gather broader insights by working with three software companies and a selection of developers with a multitude of different development tasks, as well as various different technology stacks. The same applies for the possibility of a generalization to a population of knowledge workers since our study was solely aimed at software developers. That being said, the developed monitoring application and analysis methodology may be well suited for a study with a larger set of participants as well as participants in various roles. The study should therefore be reproducible.

Chapter 6

Discussion

Our results from Chapter 5 answer two of our research questions by providing insights into the characteristics of software developers' desktop environment interactions as well as information on how traditional computer interaction data relates to visual focus. To answer our third research question, whether there is a window plague in software developers' desktop environments, we need to round up the individual results and provide some interpretation. Furthermore, we will try to set the results into perspective.

We have found software developers mostly use two monitors for their work, a primary and a secondary one. On these monitors, they have a median of 9 applications mostly for developing, browsing, emailing, and working with documents. These applications provide a median of 10 open windows and a median of 2 minimized windows. Most open windows are obscured by others so that only a median of 2 windows on the desktop are fully visible. Most of the windows and are found on the primary monitor where mostly development and debugging activities seem to take place.

When summing up time spent for browsing, emailing and document handling, we observed that our participants spend at least 39.9% of their time outside the IDE. Minelli *et al.* found their participants spend only ~8% of the time outside the IDE [11]. This difference may be attributed to the observation time frame: While they observed development sessions, we observed the complete work day of the developer. Our calculated time spent outside the IDE corroborates other research by Ammann *et al.*, that states developers spend 39.8% of their time outside the IDE. Regarding the performed activities, our results align roughly with other research as well [16].

The number of open windows that we observed matches other findings [15], even for participants without software development background [21]. The number of visible windows does also match up with other research to some extent [21]. There is research stating generally higher average numbers of open windows (as opposed to median), such as an average of 16.68 simultaneously open IDE windows [3]. Although our findings also corroborate these average numbers, they seem less meaningful with our data due to the outliers with disproportionately large numbers of open windows.

The minor deviations in the number of open windows stated by different studies may be partially explained by the discrepancy between the used applications of the participants. We have found different applications to be a major factor accounting for the large differences between our participants (see Figure 5.1). We have found some, but not all of the participants exhibited a high number of open windows. In general, the high number of open windows may be a first indication towards the existence of a window plague, especially compared with much lower number of visible windows. This divide leads us to believe that there may be some windows that get opened for single or short-time use but do not get closed afterwards. These windows are then abandoned and forgotten in the background of the desktop environment.

Over the work day, we found the number of open windows to grow (see Figure 5.2) by an average of 4 windows. A similar observation was made by other researchers for windows inside the IDE [3]. They attributed this observation towards the presence of a window plague. After deeper inspection, we were unable to find another explanation or need for these additional windows. The growth of windows over time unavoidably leads to a window plague at some point. Therefore, we agree with this attribution even outside the IDE. We estimate this to be the strongest evidence of an existing window plague in our dataset besides the high number of open windows in general. Yet, we could neither observe this phenomenon in all participants, nor was it consistently exhibited.

Over longer time periods on the other hand, we found the number of open windows for a single developer stayed more or less stable (but with notable differences between the individual developers). This might be linked to the fact that most software developers rarely turn off their computers and therefore have a more constant number of open windows. Furthermore, we observed developers cleaning up the desktop environment with rather individual strategies and frequencies. All observed desktop environments were cleaned up at some point and there were developers who stated they would cleanup regularly. One of our participants in fact stated, that he/she closes windows when he/she feels too much time is lost looking for the next window. However, a majority of developers stated that they would almost never close windows proactively. They added: Windows would only get closed for exceptional reasons, such as to free memory, for updates, or before they leave the computer for a long time. These individual cleanup strategies may also have an impact on the number of open windows and the severeness of a window plague. Moreover, these findings suggest that every developer has a personal number of open windows that he or she is comfortable with, which in turn would indicate that a window plague is not a general problem, but rather individual.

Desktop environment cleanup strategies were observed before by other researchers especially in the IDE [3, 10]. They similarly found developers were sometimes not willing to close open windows and learned from their participants that many open windows decrease development efficiency [3]. Although we have heard a similar statement from a single participant as well, many participants stated that a high number of open windows is not an issue to them. Overall, this is further evidence that a window plague may be an individual problem that does not affect every developer.

We looked at the desktop environment interactions and observed that windows were rarely moved or re-sized. Minelli *et al.* stated that 14% of a developer's time is spent on rearranging the UI of the IDE and give examples such as re-sizing or dragging windows [11]. This is not contradictory since we did not capture the time span of interactions but rather the events, and therefore make no statement of the time spent in UI Interactions. Moreover, the number provided by Minelli *et al.* may also include activations (switches) and other UI Interactions.

Most of the observed desktop environment interactions were switches between different windows (window activations). Specifically, many short term switches were observed, and windows were only active for a few seconds at the time. These results corroborated other research with similar results [1, 21, 15]. We also corroborate the observed activity patterns in window switches that other researchers have observed [1]. Other researchers had before found a high number of switches to negatively impact focus and productivity[2, 8, 16, 11]. Roethlisberger *et al.* have observed similar behavior inside the IDE and associated it to the window plague [3]. Although we believe the observation may be an indication of reduced focus and productivity, we have no evidence to attribute this behavior to the window plague. We do not, since we could not confirm a correlation between the number of open windows and the number of performed switches between these windows. Other researchers have assumed a relationship may exist due to the loss of overview and navigation efficiency associated with a large number of open windows [3].

Regarding navigation efficiency, only five of the twelve interviewed developers stated that sometimes or rarely they lose time or focus when looking for the next window. The other seven developers did not have this feeling. One participant even mentioned that his/her surroundings, such as colleges are far more distracting to him than anything in the desktop environment. Since we do not have quantifiable data on this, we were not able to verify the statements. However prominent these window switching issues may be, we can conclude that it is not prominent enough to distract most developers.

Using eye tracking data, we found that visual focus is shifted frequently, and we list the most performed switches of visual attention. Furthermore, we found that, out of all fixations, 79% were directed at the active window. Rather often, developers seem to be looking away from development applications to an instant messenger. This observation could be related to interruptions in the work flow by notifications and should be further investigated. There is other research stating the active window is only fixated for 18.75% of the time the window was active [15]. Due to the single monitor limitation, we refrain from presenting such a time based percentage value. However, based on the fixations we could observe, this result does not appear to match. We looked further into the cause and found their monitoring tool returns mostly differing ids for active window and the fixations on the same window (child window handles vs. top-level window handles). Given this situation, it is likely that the matching often failed and therefore caused the difference in the presented numbers.

In our endeavor to compare traditionally collected user input data to eye tracking data, we found the eye often tracks the cursor and that there is a correlation between the cursor input and the visual focus input. However, this does not mean the cursor and the fixation data provide the same data for a given moment in time. Overall, we found the eye tracker provides additional insights compared to the traditional user interaction data. Examples include user presence while no computer interaction is performed or insights on fixations, such as fixated windows that differ from the active window. These findings line up with the findings made by Kevic *et al.*, who stated that eye-tracking data captures substantially more and different aspects of developers interactions [31].

Overall, we observed a number of indicators of the window plague in the desktop environment that leads us to believe there is, in fact, an existing window plague that is not only restricted to the IDE. Such indicators include the observed number of visible windows, open windows, or window growth over the day. That being said, we were unable to confirm other indicators of a window plague like the number of switches. Although we found most of the developers do not close windows proactively, we can state that not every developer may be affected by a window plague; most developers do not identify the window plague as a problem. Furthermore, software developers seem in general pleased with their window-based desktop environments. Nonetheless, we see room for developer support in the desktop environment, especially concerning the high number of open windows, switches, and the desktop cleanup strategies. The provided insights may aid in the journey towards fostering developers desktop environment interactions.

Chapter 7

Future Work

The developed monitoring application provided good insight into the participants' desktop environment interactions. Nonetheless, there is a multitude of possible improvements to the monitoring application:

The biggest shortcoming of the application is the single monitor restricted eye tracking. We investigated possible solutions extensively and found none of the currently available to be viable. In the future, new eye tracking solutions may overcome the problem and therefore provide better fixation data. Furthermore, our monitoring application collects only limited information about tabs, virtual desktops or the use of newer Windows features like the timeline. The latter was not yet available as of the development of the application. Some improvements in the internal database structure would simplify the analysis, such as not only linking fixations with windows but also with desktop snapshots. Also, no information is about other devices used in collaboration or extension of the desktop environment was collected.

Concerning the analysis: Due to time constrictions, not every direction was explored so we are confident the collected data will provide many more valuable insights. Therefore, it would be fruitful to extend the analysis. For instance, to calculate a percent value for the visibility of a window that corresponds to the area visible to the developer. Using such calculations, more precise statements could be made compared to the current results which only consider fully visible windows.

Concerning the study: Since we only observed a small number of developers, we plan to extend the study to more developers to get more precise results and eliminate bias. This could allow us to present results that are in fact generalizable to a bigger subset of the population of software developers.

Our main motivation is to help developers to stay more focused and productive in their daily work. With this goal in mind, we can envision multiple approaches to support developers in the future based on our collected data. First of all, since we found the window plague to exist outside the IDE, we envision a solution that may cure those who are affected. Potential solutions may be inspired by the existing solutions used within IDEs [3, 4]. The solution should be able to detect unneeded windows and act on the information, such as indicating these windows to the developer without disrupting his work. As a starting point for such a detection mechanism, the solution would operate on interaction data as collected by our monitoring application.

Further potential ideas were formed from the insights provided by our participants. This includes an application that turns off all notifications during the time a user is highly active with an interruptibility measure based on computer interaction data similar to other research [32]. Other, more general ideas, include support of better integration and interoperability between external devices, such as smartphones and the desktop environment.

Chapter 8

Conclusion

The activities of software developers are the subject of research from various perspectives.

In this paper, we present new insights into software developers' desktop environment. We developed a monitoring application capable of capturing desktop interactions and eye tracking data. Furthermore, we implemented a processing and analysis methodology. We then deployed the application to 12 professional software developers from three different companies and conducted semi-structured interviews with all participants. In doing so, we were able to collect a dataset containing a combined total of 195 days consisting of hundreds of thousands of desktop environment snapshots and millions of windows, mouse positions, and fixations.

Using the collected data, we painted a high-level picture of software developers' desktop environments and presented their desktop environment interactions. Furthermore, we compared the eye tracker data to traditional computer interaction data and found it provides insights into different aspects, such as user presence and visual focus.

Our main findings constitute: From our eye tracking, we know 79% of all observed fixations were directed at the active window. Half of the desktop environments had 10 or more open windows but with only two fully visible. We found the number of windows to grow over the day and concluded there must be windows that do not get closed after use. Moreover, we observed desktop environments being cleaned up at various points limiting growth in the long term. Our conclusion about the growing number of windows was substantiated by interviews where we learned most developers do not proactively close windows. These observations lead us to believe, that there is in fact an existing window plague outside the IDE, although it does not affect every desktop environment. In other words: Software developers sometimes have overcrowded desktop environments with many unused windows.

We want to rise awareness of unused windows in the of desktop environment and we hope our work may inspire new solutions to increase developers' focus and productivity in the near future.

Bibliography

[1] 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 Transactions on Software Engineering*, 43(12):1178–1193, Dec 2017.

[2] H. Sanchez, R. Robbes, and V. M. Gonzalez. An empirical study of work fragmentation in software evolution tasks. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 251–260, March 2015.

[3] D. Roethlisberger, O. Nierstrasz, and S. Ducasse. Autumn leaves: Curing the window plague in ides. In *2009 16th Working Conference on Reverse Engineering*, pages 237–246, Oct 2009.

[4] R. Minelli, A. Mocci, and M. Lanza. The plague doctor: A promising cure for the window plague. In *2015 IEEE 23rd International Conference on Program Comprehension (ICPC)*, volume 00, pages 182–185, May 2015.

[5] Janice Singer, Timothy Lethbridge, Norman Vinson, and Nicolas Anquetil. An examination of software engineering work practices. In *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON '97, pages 21–. IBM Press, 1997.

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

[7] Gloria Mark, Victor M. Gonzalez, and Justin Harris. No task left behind?: Examining the nature of fragmented work. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '05, pages 321–330, New York, NY, USA, 2005. ACM.

[8] Gloria Mark, Daniela Gudith, and Ulrich Klocke. The cost of interrupted work: More speed and stress. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 107–110, New York, NY, USA, 2008. ACM.

[9] G. C. Murphy, M. Kersten, and L. Findlater. How are java software developers using the elipse ide? *IEEE Software*, 23(4):76–83, July 2006.

[10] R. Minelli, A. Mocci, M. Lanza, and L. Baracchi. Visualizing developer interactions. In *2014 Second IEEE Working Conference on Software Visualization (VISSOFT)*, volume 00, pages 147–156, Sept. 2014.

[11] R. Minelli, A. Mocci, and M. Lanza. I know what you did last summer - an investigation of how developers spend their time. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 25–35, May 2015.

[12] S. Amann, S. Proksch, S. Nadi, and M. Mezini. A study of visual studio usage in practice. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 124–134, March 2016.

[13] David M. Hilbert and David F. Redmiles. Extracting usability information from user interface events. *ACM Comput. Surv.*, 32(4):384–421, December 2000.

[14] Anton N. Dragunov, Thomas G. Dietterich, Kevin Johnsrude, Matthew McLaughlin, Lida Li, and Jonathan L. Herlocker. Tasktracer: A desktop environment to support multi-tasking knowledge workers. In *Proceedings of the 10th International Conference on Intelligent User Interfaces*, IUI '05, pages 75–82, New York, NY, USA, 2005. ACM.

[15] Nick Bradley and Felix Grund. Characterizing knowledgeworkers' desktop interactions. Unpublished paper, 2017.

[16] André N. Meyer, Thomas Fritz, Gail C. Murphy, and Thomas Zimmermann. Software developers' perceptions of productivity. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 19–29, New York, NY, USA, 2014. ACM.

[17] K. D. Fenstermacher and M. Ginsburg. A lightweight framework for cross-application user monitoring. *Computer*, 35(3):51–59, March 2002.

[18] Simone Stumpf, Xinlong Bao, Anton Dragunov, Thomas G. Dietterich, Jon Herlockel, Kevin Johnsrude, Lida Li, and JianQiang Shen. The tasktracer system. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 4*, AAAI'05, pages 1712–1713. AAAI Press, 2005.

[19] Simone Stumpf, Xinlong Bao, Anton Dragunov, Thomas G. Dietterich, Jon Herlocker, Kevin Johnsrude, Lida Li, and Jianqiang Shen. Predicting user tasks : I know what you ' re doing ! 2005.

[20] Xinlong Bao, Jonathan L. Herlocker, and Thomas G. Dietterich. Fewer clicks and less frustration: Reducing the cost of reaching the right folder. In *Proceedings of the 11th International Conference on Intelligent User Interfaces*, IUI '06, pages 178–185, New York, NY, USA, 2006. ACM.

[21] Dugald Ralph Hutchings, Greg Smith, Brian Meyers, Mary Czerwinski, and George Robertson. Display space usage and window management operation comparisons between single monitor and multiple monitor users. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, pages 32–39, New York, NY, USA, 2004. ACM.

[22] Mik Kersten and Gail C. Murphy. Mylar: A degree-of-interest model for ides. In *Proceedings of the 4th International Conference on Aspect-oriented Software Development*, AOSD '05, pages 159–168, New York, NY, USA, 2005. ACM.

[23] M. E. Crosby and J. Stelovsky. How do we read algorithms? a case study. *Computer*, 23(1):25–35, January 1990.

[24] Roman Bednarik. Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations. *Int. J. Hum.-Comput. Stud.*, 70(2):143–155, February 2012.

[25] Shehnaaz Yusuf, Huzefa H. Kagdi, and Jonathan I. Maletic. Assessing the comprehension of uml class diagrams via eye tracking. *15th IEEE International Conference on Program Comprehension (ICPC '07)*, pages 113–122, 2007.

[26] Bonita Sharif and Jonathan I. Maletic. An eye tracking study on the effects of layout in understanding the role of design patterns. *2010 IEEE International Conference on Software Maintenance*, pages 1–10, 2010.

[27] Bonita Sharif and Jonathan I. Maletic. An eye tracking study on camelcase and underscore identifier styles. In *Proceedings of the 2010 IEEE 18th International Conference on Program Comprehension*, ICPC '10, pages 196–205, Washington, DC, USA, 2010. IEEE Computer Society.

[28] Benoít De Smet, Lorent Lempereur, Zohreh Sharafi, Yann-Gaël Guéhéneuc, Giuliano Antoniol, and Naji Habra. Taupe: Visualizing and analyzing eye-tracking data. *Sci. Comput. Program.*, 79:260–278, January 2014.

[29] Bonita Sharif and Huzefa Kagdi. On the use of eye tracking in software traceability. In *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '11, pages 67–70, New York, NY, USA, 2011. ACM.

[30] Thomas Fritz, Andrew Begel, Sebastian C. Müller, Serap Yigit-Elliott, and Manuela Züger. Using psycho-physiological measures to assess task difficulty in software development. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 402–413, New York, NY, USA, 2014. ACM.

[31] K. Kevic, B.M. Walters, T.R. Shaffer, B. Sharif, D.C. Shepherd, and T. Fritz. Eye gaze and interaction contexts for change tasks observations and potential. *J. Syst. Softw.*, 128(C):252–266, June 2017.

[32] Manuela Züger, Christopher Corley, André N. Meyer, Boyang Li, Thomas Fritz, David Shepherd, Vinay Augustine, Patrick Francis, Nicholas Kraft, and Will Snipes. Reducing interruptions at work: A large-scale field study of flowlight. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 61–72, New York, NY, USA, 2017. ACM.

Interview Protocol

The interview protocol of our semi-structured interview conducted with 12 participants from three different companies is presented in the following.

Hi ... ,
Thank you very much for participating in our study. In the last weeks, you have been observed by our monitoring application. With this interview, we aim to get a bit more insights into how developers use their desktop environment beyond what the software can provide. At first, we need to define some words to ensure we're talking about the same terminology.

Definitions

- *Window* is what you typically find in the operating system. It separates an area of the computer screen and acts as the user interface for an application. We exclude tab windows from this definition and will talk about tabs.
- *Tabs* you find typically in your browser application. The tabs allow to bundle the contents of multiple documents that would typically use multiple windows into a single window and allows switching between these contents.
- *Notifications* are anything that delivers information based on an event without a request from the user. We do not restrict this to the notifications provided Microsoft Windows.

General

1. What best describes your primary work area?
   [*Development, Test, Project Management, Other Engineer, Other Non-Engineer*]
2. Which of the following best describes your role?
   [*Individual Contributor, Lead, Architect, Manager, Executive, Other*]
3. How many years of software development experience do you have in total?
4. How many years of professional software development experience do you have?

Desktop Environment Use

5. How many monitors do you use at your work?
6. How much time do you spend on each?
   [*the same, more on the primary, lot more on the primary, rarely using the others*]

7. How do you split your work onto these monitors?
8. What number of windows/applications do you usually use simultaneously?

   [*1-3, 4-6, 7-9, 10+*]
9. What number of browser tabs are usually opened simultaneously?

   [*1-4, 5-9, 10+, 20+, 30+*]
10. What are your typical use cases / workflows for simultaneously open windows?
11. What is your strategy to closing open windows/tabs?

    [*e.g: When it gets too cluttered, I close them immediately after I don't need them anymore, After a task, Before a new task, I close everything before a break, I close everything in the evening when I leave, I never close anything, a mixture of the above (explain)*]
12. Why do you choose this strategy?

Finding Windows/Tabs

13. Do you lose time when looking for the next window or tab?

    [*never, rarely, sometimes, often, regularly*]
14. Do you ever lose focus/track of your task when searching for a window? How common is it to lose focus/track of your task at hand?

    [*never, rarely, sometimes, often, regularly*]
15. Why do you sometimes lose time/focus/track to find the next window or tab?

Distractions

16. How easy/ often ... are you distracted by open windows/tabs/notifications that do not belong to your current task?

    [*never, rarely, sometimes, often, regularly*]
17. Regarding the window environment: what do you find particularly distracting?

Support

18. Is there anything that you think could be better in your windows environment?
19. Do you have any ideas on how to help you with your workflow?
20. If we were to develop an approach to limit distractions or help you find the relevant window/tab/application faster, how would we go on to do that?

Study related

21. Was the past week a usual work-week? If not, why?
22. Did the tracker influence you? And if so, in what way?

Contents of the CD-ROM

- **Abstract.txt**
  The plain text abstract of this thesis in the English version.
- **Zusfsg.txt**
  The plain text abstract of this thesis in the German version.
- **MasterThesis.pdf**
  Copy of this thesis as PDF.
- **Interview.pdf**
  The interview protocol.
- **SourceCode.zip**
  The source code of the monitoring software.
- **AnalysisNotebook.Rmd**
  The R Notebook of the Desktop Environment Analysis
- **VisualAttentionNotebook.Rmd**
  The R Notebook of the Analysis of the Visual Attention

The collected data and analysis results (such as outputs of the R notebooks) are not included on the CD-ROM due to their privacy sensitive nature.