University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Mateusz Bajorek
Student no. 385122

Analysis and visualization of programmers' activities during their workday

Master's thesis
in COMPUTER SCIENCE

Supervisor:
dr Robert Dąbrowski
Institute of Informatics

Warsaw, December 2022

Abstract

Similarly to another professions, programmers create their own working style. It can be characterized by different phases of the day, splitting tasks into a few chunks, taking breaks, or focusing on a few projects at the same time. In this thesis, I analyze if there exist links between programmers preferences and the spectre of their duties, for example, if it is true that front-end programmers work usually in the evenings. For this purpose, I conduct a research scheme on a sample of programmers with the help of a self-created software tool for gathering anonymized data (including the number of projects, time of activities and programming languages). The end result of this thesis comprises of the conclusions from the analysis of the aforementioned data with included visualizations. The tool used in the thesis is shared publicly for the willing actors to independently verify my end results by re-running the experiments or running new ones to gather new information, for instance, about potential improvements in programmers efficiency.

Keywords

programming process analysis, programming process visualization, keystroke dynamics

Thesis domain (Socrates-Erasmus subject area codes)

11.3 Informatics, Computer Science

Subject classification

D. Software

Tytuł pracy w języku polskim

Analiza i wizualizacja aktywności programistów w ramach ich dnia pracy

Acknowledgements

First of all, I would like to thank my supervisor PhD Robert Dąbrowski for expert guidance during the "Software Optimization, Visualization and Analysis" (SOVA) seminars. His help in the university administrative procedures involving this thesis, overall assistance and patience in overcoming various obstacles by me was invaluable.

Another lecturer present during the seminars was Prof. Krzysztof Stencel who provided me with essential advice. I greatly appreciate his willingness to share critical insight related to my thesis and the whole defence process.

Apart from the leaders of the seminars, I must mention my colleagues who were attending my presentations. This was the time when we shared our remarks on each other's work. My thesis would not be the same without their input during these discussions.

From my colleagues, I must single out and thank Małgorzata E. Z. Galińska who was the first to test my VS Code plugin and assisted me with debugging it.

I thank my former employers and dear friends Mathias Åsberg (CEO NewNative) and Radosław Dembkowski (CEO RadCode) for their efforts at helping me gather the data.

I applaud everyone who decided to share their data to proceed with my research project. It would be a cliché to say that my thesis would not exist without them.

I am eternally grateful for the Faculty of Mathematics, Mechanics and Informatics of the University of Warsaw for giving me the chance to study Computer Science and conduct my research there. Every year spent at the Faculty was an inimitable adventure for me.

Last but not least, my thanks go to my friends for their indispensable presence and my family for their immense patience and unwavering support both emotional, and financial.

Warsaw, December 2022
Mateusz Bajorek

Contents

Chapter 1

Introduction

Programmers job focuses on developing computer software through all phases including design, testing and maintenance. These processes are executed according to the principles of performance, reliability and security [3]. With the industry maturing, upholding these rules in ever-expanding projects have become crucial to fulfill growing needs stated i.a. in software level agreements (SLAs). One of the answers to that have been the agile software development [5], where actions like joining regularly scheduled team meetings, contacting clients or attending conferences have been universalized. Nevertheless, writing code has remained a key part of programmers job for the majority of them. In my study, I focus on analyzing and visualizing the exact process of writing software in an integrated development environment (IDE), largely omitting the other aforementioned aspects of being a programmer.

Research in the relevant areas differs depending on the context that is industrial or educational. Tracking of programmers time happens to be an essential part of the job in the IT industry. The reports provided by programmers are the basis for calculating a salary or even deciding the future of an employee by the management. For many programmers it may be hard to determine which activities constituted billing time or if different activities should be treated differently. In my own experience, hard to fix bugs or complicated issues solving which took many trials and errors happened to be particularly hard when it comes to calculating the exact amount of billing hours it took me and how to prove this reported effort to my employer. Those obstacles constituted my initial desire to explore the domain of programming process analysis and visualization. Nowadays there exist tools that measure the time automatically and compile the reports with almost no additional human effort.

In the industry there is less focus on the detailed analysis of the process of writing the code and more on the activities done during the whole workday. Process of writing the code is more of a concern for educational research and IDE development companies like JetBrains who want to know how users interact with their products and how to make them more ergonomic. In educational research people try to evaluate students based on their typing behavior. Apart from that, it's possible to authenticate users based on their typing characteristics.

The thesis contains 5 chapters and appendices. In the chapter 2 there are studied basic concepts and related works both in analysis and in visualization techniques. The chapter 3 describes the tool used to gather and analyse the data. In the chapter 4 the results are presented and in the next one 5 they are discussed. The last two chapters conclude the whole thesis with suggestions for future work 6 and provide links to the additional published materials 7. I clarify that the pronoun "I" used in this thesis refers to its author.

Chapter 2

Background

2.1. Basic concepts with their applications

Throughout the thesis I use terms distinctive to biology, software engineering or IT industry. I have decided to highlight some of them due to their importance and ubiquity in my thesis or in the related works. Apart from the essential definitions, I provide the applications together with my motives to utilize this terminology.

Definition 2.1.1 Circadian rhythms *are a set of physiological changes that follow a cycle of roughly 24 hours. These processes are governed by a biochemical clock in the brain that receives the fluctuating light during the day and at night. They fundamentally influence person's activity levels at different hours. [1]*

Circadian rhythms enable humans to follow day cycles. It is an important concept in my thesis since I mainly consider programmers work on the basis of separate days. Within a single day, there are expected differences of behavior between programmers, partially explainable by a notion of chronotypes.

Definition 2.1.2 Chronotype *is a natural tendency of a person toward increased or decreased activity at certain parts of the day. It is closely coupled with human's circadian rhythms. [2]*

The theory of individual's chronotype is a major reason why some developers exhibit greater activity during earlier parts of the day while the others during the later ones. Such a phenomenon occurs obviously within the confines of social norms. Programmers of late chronotypes (the so-called "night owls") are subject to the same common working hours as the ones of early chronotypes (the so-called "morning larks"). Thus, the true extent of individual's chronotype can be further studied by analyzing weekend activity or volunteer work on open-source projects (which may, in fact, still be influenced by external factors). Nevertheless, chronotypes remain a vital factor in determining programmers productivity peaks and slumps which are in the focus of my thesis.

Definition 2.1.3 Time tracking *refers to a company policy regarding logging time of its employees. The reports compiled in this process present a summary of time spent on each given task by an employee. [4]*

In ordinary company operations chronotypes are not a matter of general concern. What undoubtedly matters is the eventual completion of tasks assigned to each programmer. In

order to secure this goal, the systems of time tracking are introduced to software development teams. This way superiors can estimate time required to achieve milestones and compensate employees fairly according to their performance. Several techniques of time tracking are comparable to the ones used to investigate chronotypes. My belief is that integrating the idea of a chronotype into time tracking programs may contribute to improved deadline estimates and more accurate assessment of programmers conduct.

Definition 2.1.4 Keystroke dynamics *is a research area concerning the rhythm of typing. It generally involves recording person's keystrokes with their timestamps in order for the gathered data to be further analyzed. [9]*

The domain of keystroke dynamics is imperative for examining human behaviour at a computer keyboard. It can be utilized to study a wide array of subjects including user authentication [34] and programming performance [6, 7]. Since capturing and inspecting keystrokes are my main research methods, findings pertinent to keystroke dynamics provide an indispensable help in my thesis.

Definition 2.1.5 Programming process *relates to the way how a particular programmer develops a given piece of software. This process can be observed by tracking the programmer's actions linked to typing the source code on a keyboard. [13]*

Tracing programming process through the means of keystrokes capturing is my main objective. It is potentially a great observation point for chronotypes. It is also a point of concern for time tracking. Thus, it can be seen as a handy concept that clusters together multiple ideas mentioned earlier.

Definition 2.1.6 Granularity *of the data refers to the number of recorded data points per unit of time.*

In short, data granularity is simply the relative density of data points. Greater granularity (denser data) allows for potentially more extensive studies, while lesser granularity (sparser data) saves computer resources. This and other dilemmas regarding the data granularity aspect in research studies are examined in literature [24].

Definition 2.1.7 Code snapshot *is a saved state of programmer's work at a specific point in time. It typically contains a copy of all the source code that was written until the time of capturing the snapshot. [25]*

Storing a sequence of code snapshots gives an exceptional insight into the programming process. It may be of different granularities with keystroke level comprising dense data points and git commits - sparse ones. In the literature code snapshots may also relate to the gathered metadata (like timestamps of user actions). Thus, it can be said that my system in fact gathers subsequent code snapshots.

My work is focused on analyzing chronotypes of different programmers and finding relations between chronotypes and various factors like programming languages used. The chronotypes may exhibit different characteristics like the frequency and length of breaks, or amplitude of activity peaks. Apart from the chronotypes, I am interested in the particular actions themselves, especially the difference in deletes and additions. To see if there is some kind of variation in this regard between programmers or clusters of programmers within a given language.

2.2. Tracking programmers time in the IT industry

In many areas of the IT industry, programmers salaries are the most significant expense and not the computation time (electricity bills, server maintenance etc.). In such a case, effective management of these human resources becomes a crucial task for keeping finances of a company in check. That is why it is essential to know exactly how many hours need to be spent for each programmer and on which tasks. For this purpose many tools were created to generate relevant reports for the employer's oversight on the company and for the employees themselves to motivate them to earnest effort.

The tools like Trello, Jira, YouTrack allow for manual time tracking. The programmers are required to manually mark their progress. They are tightly coupled with software development methodologies (established ways of navigation through each phase of the software development [28]). Handling agile frameworks such as Kanban and Scrum is especially important for these tools.

Nevertheless, there are also tools which help with automated time tracking. Such tools include Timely and Tempo products. They track user actions and evaluate them using different techniques (including those based on machine learning) to label them accordingly. This way the users can save the time on filling out the reports, especially thanks to built-in integrations with Jira etc. Another interesting solution is to integrate dashboards directly into an IDE. Notable examples of that approach include WakaTime and CodeTime. The visualizations and statistics that these tools conveniently generate for the IDE users are comparable to those available in my tool. However, one of my main premises was to ensure the availability and security of the gathered data adhering exactly to my specification. Thus, I have decided to built my own tool according to the aforementioned principles.

The agile software development methodologies mentioned above (Kanban and Scrum) utilize special dashboards with clearly described tasks for programmers. They represent an interesting way of visualizing activities of developers. My thesis does not cover detailed labeling of activities which is required by those techniques. Nevertheless, it is a concept worth noting for potential further research. The emergence of the automated time tracking tools (e.g. Timely and Tempo products) show that the IT industry is attentive to the developments in the area of sophisticated analyzing and visualizing of programmers' activities. I hope my thesis also gives some insight into this matter.

2.3. Usage of programmers data in research

Analysis of gathered programmers data is a vibrant area of research. I would like to make a quick overview of such research, especially this most relevant to my thesis.

Zavgorodniaia et. al. [11] collected keystroke data from two groups of introductory programming students (European and American) using a custom IDE. This data allowed the research team to search for the activity patterns associated with distinct chronotypes. They identified four chronotypes isolated by Putilov et. al. [29] that is morning, napper, afternoon, and evening. The difference in scores was the most noticeable within the US context, where morning persons achieved the highest exam scores. Comparing the contexts, it was observed that European participants usually started their work earlier than their American counterparts. Researchers stipulated that the coincidence of their results with the prior published research does not prove that students' circadian rhythms and diurnal preferences are the sole reason for their activity patterns. Social factors like rigid schedules or external duties might have impacted the conclusions since the surveys for such factors were not taken as part of the

experiments.

Claes et. al. [19] examined programmers work patterns by analyzing version control commits from multiple software projects. The researchers found that typically developers follow the standard circadian rhythm where effort is concentrated in the morning and in the afternoon. Unsurprisingly, they observed characteristic dips in activity around the lunch time (11 or 12 o'clock depending on the project) and much bigger ones during the weekends. The myth of programmers being night owls was not confirmed since according to the data the median 8-hour working day is placed between 10 and 18 o'clock. In addition, the data showed that two thirds of developers work primarily within the office hours. The researchers found some evidence that activity outside the normal office hours is correlated with non-paid (hobby) work.

Robinson [20] examined in his article the data from Stack Overflow about visits to the site. Stack Overflow is a popular platform where developers ask and answer technical questions. The author of the article decided to inspect developers programming behavior assuming a strong link between Stack Overflow and programming activity. Not surprisingly, the peak activity was observed during the weekdays within the standard 9-to-5 time period (adjusted for local time zones). Moreover, a dip in traffic was observable at 12 p.m. coinciding with a typical lunch hour. On weekends such a dip did not exist. Across several programming languages there were noticed slight differences in hourly trends, that is some of the languages were e.g. associated with statistically earlier or later times of activity. The author also highlighted the technologies that were relatively most used within or outside the typical working hours (9-to-5). Multiple Microsoft (a major IT company) products demonstrated high percentages of usage within the working hours, while Haskell (a functional programming language), Firebase (an application development platform) and Unity (a game engine) outside of them. In addition to that, there was analyzed usage in the morning and the afternoon, and found a general correlation between usage within working hours and in the morning. At the end, it was discovered that cities known for remote work exhibit a bigger number of late chronotypes comparing to traditional software/business centers with locally employed programmers.

Silge [21] explored the diversity of questions that are posted on weekdays and weekends. She noticed that some languages become more popular during weekends (with regards to the relative frequency on weekdays vs on the weekend). There are also trends of diversion in these statistics throughout time which may suggest maturing of certain technologies (moving activity from weekends to weekdays) or going more mainstream with hobbyists (moving activity from weekdays to weekends).

2.4. Tools for visualizing the programming process

Visualization of software is a vigorously studied area [23, 22]. It concerns mainly representations of software structure, behavior and evolution [27]. In my thesis I am mostly interested in the last point with emphasis on visualization of the programming process by analyzing keystrokes. The different takes on this matter are presented below.

Balzuweit et. al. [17] developed a prototype tool named **SnapViz** which aims to standardize the systems for displaying snapshot data gathered from programming students. The tool consumes timestamped data points containing information about a student, a task, the student's score achieved on the task with the relevant code snapshot, and a label (e.g. an error message). Thanks to the data, a plot is generated with time and date axes, that presents how individual's solutions evolved in time. This way it can be observed how many attempts

a student took and if the score increased gradually or with a spike. The authors of the article hoped to spur discussion and they actually succeeded judging by references appearing in some of the papers mentioned below.

Shrestha et. al. [16] presented a tool called **CodeProcess** for visualizing the programming process 2.1.5. The main point of the tool is to show the order in which the source code was written. The tool allows to see the keystroke data in a plot arranged from each character added. The authors conducted a study to evaluate the usefulness of the tool for educators. They came with multiple potential areas for the tool including grading assistance, determining student's approach to the problem (top-down or bottom-up), detecting plagiarism, or finding struggling students in need of support. Apart from teachers, students could use the tool to reflect on themselves to see different approaches in overcoming potential obstacles. The authors state that the process of analyzing the plots could be enhanced by ML to enable automated reasoning (e.g. for plagiarism screening). Apart from education, the ideas behind the tool are expected to be used in the professional context for code reviews and self-growth of programmers.

Matsuzawa et. al. [13] developed a tool named **Programming Process Visualizer** (**PPV**) based on the Personal Software Process (PSP) framework. It comprises a set of techniques to help engineers and their organizations improve their performance through a disciplined approach to measuring and analyzing their work [26]. The PPV software allows students to analyze their programming process after completing a programming task. The tool provided students with the replay ability, so they could see the animation of writing the source code. In parallel, the metrics are calculated, including number of lines, working time and other information about compilations and runs. A notable feature crucial for the PSP is the ability to provide labels for the subtasks, that is pieces of code, for what students actually attempted to achieve by writing them. This way students are able to estimate the time they thought it would take them to complete an assignment and compare it with the real time span. The researchers tested their tool empirically on the students and it gave them promising results with the students providing positive responses.

Biehl et. al. [14] created a tool called **FASTDash** to enhance team awareness in software development projects. They intended to achieve that by deploying a visualization that provides information about current activities of each programmer. This way a developer can quickly learn which team members are working on each files or even exactly on which parts of code like classes or methods. The developers can even provide comments for better understanding. One of the more obvious advantages of this solution is to mitigate the risk of conflict situations. The tool was tested in an experimental environment with promising results and some developers continuing to use it afterwards.

Lyulina et. al. [15] developed a toolkit named **TaskTracker** for collecting and analyzing data of student programming process. The gathered data includes code snapshots of high granularity (even with 1 character change) and other user actions taken within the IDE based on the IntelliJ Platform. The tool allows for tracking solving a fixed number of predefined tasks in various programming languages. The authors created and published a dataset by tracking activities of 148 participants. It is interesting to see that the tool can be used to detect plagiarism similarly to one of the tools above.

Norris et. al. [18] introduced a tool called **ClockIt** for monitoring student programming practices. It logs opening and closing projects or packages, compilation and run events, as well as files deletions and changes. The gathered data is presented in summary visualizations of compilation statuses and invocation outcomes within chosen time periods. The tool provides also the activity graph with the total count of events in time and the project growth graph with the total number of lines of code and comment in subsequent days.

Minelli et. al. [47] presented **DFlow** for recording and analyzing programmer's interactions with the Pharo IDE. The vital point of DFlow is its rather unique (at least at the time of publishing) scheme of visualizing events such as navigation and edits with respect to program entities (classes, methods and attributes). It enabled graphing programmer's behavior when shifting focus to different parts of the code. In the next paper [48] Minelli et. al. expanded on this concept by focusing on developers' experiences with the UI of the Pharo IDE. They created more visualizations, mostly by mapping programmers' actions on a timeline. The intention of this study was to help developers optimize their experience with the IDE.

Di Rosa et. al. [49] published a study concerning visualizing programmers interaction data inside and outside the IDE. They achieved this by building the **Ferax** tool on top of the existing plugin for Visual Studio Code called **Tako** which records and analyzes activity within the IDE. Despite possessing many desired features, I am not using Tako for my study since its source code is not publicly shared (its GitHub repository does not contain the actual code), so making any changes is constricted. In addition, the collected data is very fine-grained, thus the amount of data that would be generated could have been troublesome for storing on participants' computers and then transferring it to my cloud storage for further analysis (see 2.1.6). Returning to Ferax, it provides many visualizations related to programmers using various kinds of applications, i.e. how much time they spent in programming, browser or social network apps. This way researchers got insight into distraction and productivity levels of their test subjects.

The tools listed above differ in the granularity of the data gathered and the scope of the code base analysed. Some of them work on the keystroke level while the other analyze git commits. There are those focusing only on one source code file and those analyzing the entire code repository. Instead of attempting to utilize one of the available technologies, I have decided to build my own tool with the chosen level of granularity and the scope of observed code files. Nevertheless, the papers above constitute a meaningful source of inspiration for my work. I note here that Zhevaho [50] prepared an overview of tools for collecting the programmers data with many of these tools containing a visualization component. I recommend this paper to anyone interested in discovering less known programming process visualizers.

Chapter 3

Methodology

3.1. Data

The data was being gathered using the IDE plugin called `MIMUW-MB-TT` 3.2 in the period between 10th October and 25th November 2022. 15 programmers participated in the research. They usually contributed information about at least a few of their working days 3.1. Majority of the data related to Python and JavaScript/TypeScript (JS/TS) languages 3.2. JavaScript (JS) and TypeScript (TS) are treated as one language later in my analysis due to their similarity (TS is a typed superset of JS) and the fact that TS is transpiled to (plain) JS [8].

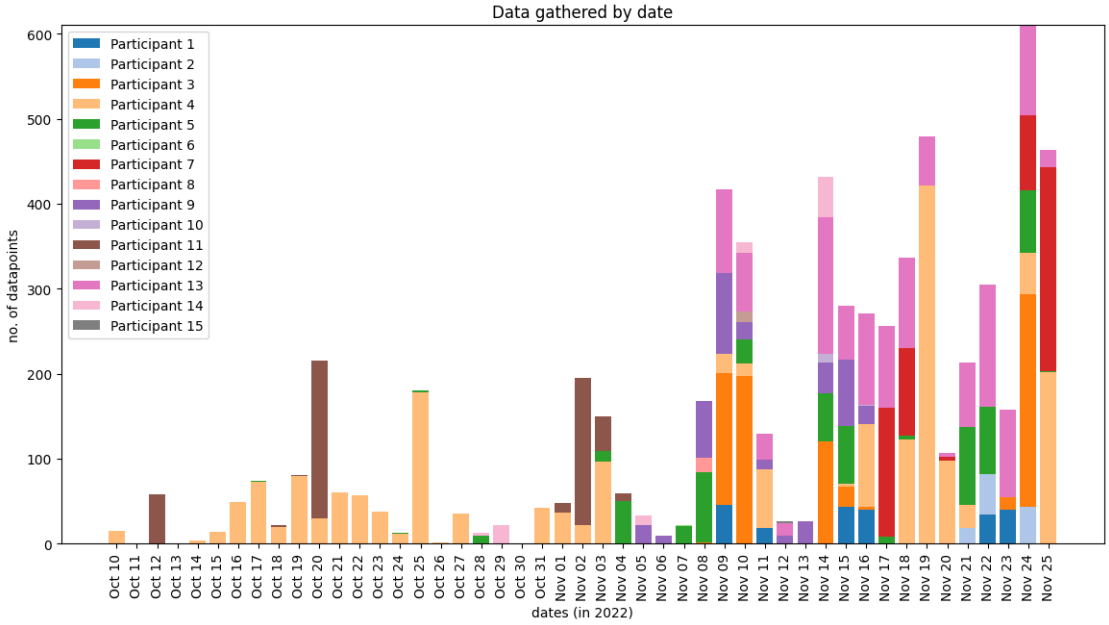

Figure 3.1: Data gathered from participants by date

3.2. Context

The data was gathered from different sources including: my acquaintances, students of my faculty reached directly and through the Facebook (a popular social media platform) group

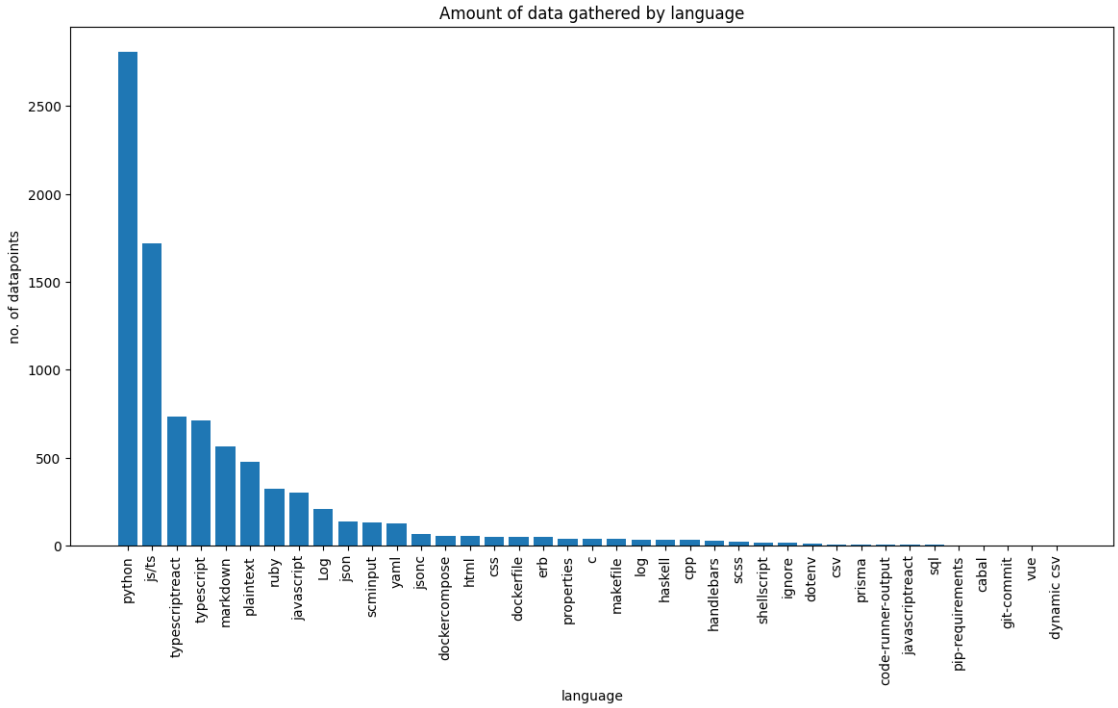

Figure 3.2: Amount of data gathered by language

consociating them, as well as my former employers who volunteered to help. Attempts at gathering data from Reddit (a popular internet forum) users failed.

I offered certain rewards to encourage participation in the research project, which included a chocolate treat (inspired by the article of Feitelson et. al. [12]), small amount donated to a charity and obviously an unrestricted access to the tool with all of its features in the current version. Research suggests that such external incentives are not particularly effective [31, 32], nevertheless it was my way of showing appreciation to the contributors without generating potential conflicts with the law. Redistributing digital goods, for instance, may be seen as a compelling way of rewarding, but it goes against EULAs (end user agreements [38]). The charity incentive could have proved quite effective especially in the current times [35]. However, my actual experiences with conducting the research for this thesis confirm the results of the aforementioned research about the ineffectiveness of such incentives.

The tool `MIMUW-MB-TT` is build on the top of Visual Studio Code 3.3 (a popular IDE-like code editing software) due to its prevalence within the programming realm [30]. Furthermore, VS Code is easily extensible with the common TypeScript/JavaScript (JavaScript is pervasive in web development and TypeScript is basically typed JavaScript) framework and I have professional experience in building web applications. My plugin and its source code are available to the general public 7.

In the end the `MIMUW-MB-TT` plugin was released on 10th October 2022 and last updated on 21st October 2022. The participants were instructed not to uninstall the plugin for one week at bare minimum or better at least for two weeks. As of 25th November 2022 the plugin was downloaded 33 times and I have received data from 15 people. Technically speaking, 15 packages of user data has been gathered and I assume that each of them comes from a different person. The tactic of reaching out directly to the faculty students and driving them to download my plugin by rewarding them with a chocolate seem to result with the most

prominent source of participants in the study.

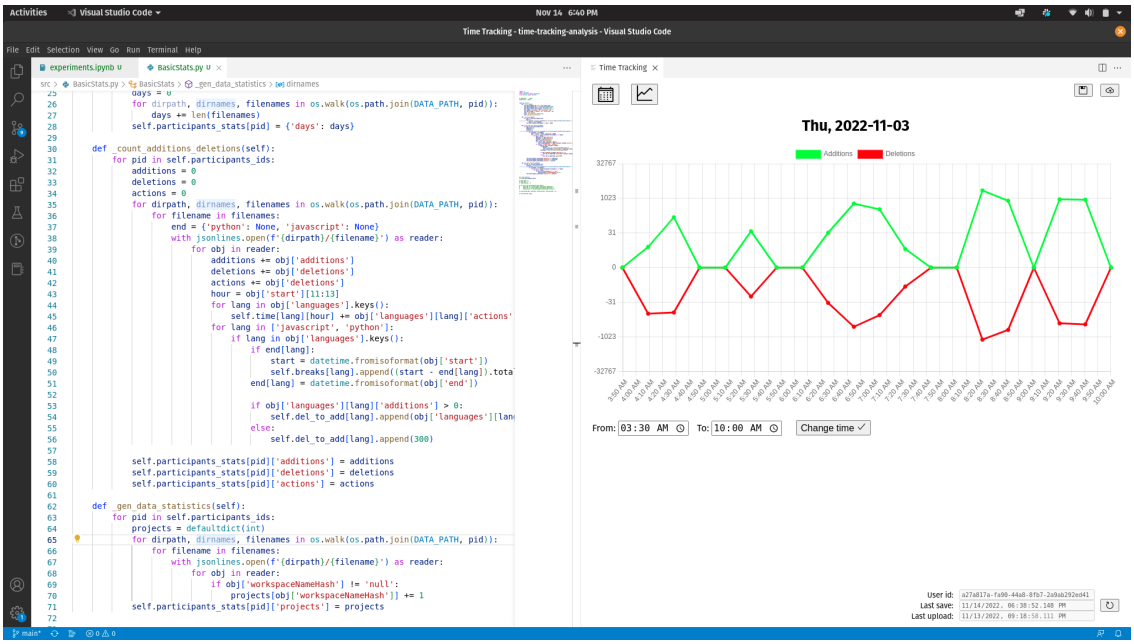

Figure 3.3: Extension embedded in Visual Studio Code

3.3. Data description

The data is stored in the file system in the `JSON Lines` (`.jsonl`) [37] format. It was popularized in recent years by OpenAI (a popular AI/ML company) in its GPT-3 API [36]. Essentially files in this format are text files where each line (seperated by newline "`\n`" characters) constitute a valid JSON value (that is why this format is sometimes also called `newline-delimited JSON`). In the particular case of my tool each of these JSON lines has a special designated type 3.4. As an example I present below a sample of the gathered data 3.5.

Let me describe each of the fields of this structure:

`additions` say how many characters (chars) were added to the files opened in the editor within the time-frame determined by the fields `start` and `end`;

`deletions` say analogously to the `additions` field about the deleted chars;

`actions` count how many actions were done on each file opened in the editor (one action may add or delete multiple chars);

`start` means the start of the interval during which the record of the data was gathered;

`end` is analogously to the `start` field the end of the interval;

`workspaceNameHash` is the SHA256 [33] hash of the workspace name in VSCode. It is done to secure the privacy of the research participants in the well established way (not to leak these names), but simultaneously to still be able to differentiate between the project names. Thanks to this, I am able to use this field to calculate the number of distinct projects of each user. If the workspace name does not exist, the value of `workspaceNameHash` is `null`;

```
{
  additions:                  INTEGER
  deletions:                  INTEGER
  actions:                    INTEGER
  start:                      STRING         // ISO 8601 date-time format
  end:                        STRING         // ISO 8601 date-time format
  workspaceNameHash:          STRING | NULL  // SHA-256 string or null
  languages: {                DICTIONARY
    [languageId: STRING]: {   DICTIONARY
      additions:              INTEGER
      deletions:              INTEGER
      actions:                INTEGER
    }
  }
}
```

Figure 3.4: The data type

`languages` is a dictionary that stores data separately for each programming language. It basically splits the fields `additions`, `deletions` and `actions` into contributions from each language. To compute this data, I have used the feature of VS Code that automatically recognizes a programming language from the file extension [10].


```
{"additions":49,"deletions":2,"actions":59,
"workspaceNameHash":"47ba341f442134c53c9cd42d8e4ab6d627ab7ca632ffdf60a432c7c6e2df9544",
"languages":{"javascript":{"additions":49,"deletions":2,"actions":59}},
"start":"2022-10-17T14:15:04.381+02:00","end":"2022-10-17T14:15:30.636+02:00"}

{"additions":105,"deletions":11,"actions":30,
"workspaceNameHash":"47ba341f442134c53c9cd42d8e4ab6d627ab7ca632ffdf60a432c7c6e2df9544",
"languages":{"javascript":{"additions":105,"deletions":11,"actions":30}},
"start":"2022-10-17T14:16:26.214+02:00","end":"2022-10-17T14:16:43.514+02:00"}

{"additions":12,"deletions":6,"actions":20,
"workspaceNameHash":"47ba341f442134c53c9cd42d8e4ab6d627ab7ca632ffdf60a432c7c6e2df9544",
"languages":{"javascript":{"additions":12,"deletions":6,"actions":20}},
"start":"2022-10-17T14:16:59.027+02:00","end":"2022-10-17T14:17:28.828+02:00"}
```


Figure 3.5: Fragment of an example data file with 3 data points (normally there is one data point per line, but here the lines are wrapped to fit them on the page)

3.4. Premise

I utilize the 30 seconds long intervals in order to simultaneously provide dense enough granularity and prevent user identification which may happen if intervals were too short. According to the article by Leinonen et. al. [34] the identification stops being feasible when keystrokes are segregated into buckets of around 300 milliseconds (and more). My buckets (intervals) are much bigger and I do not save information about concrete keys pressed - I store only cumulative data (mainly added and deleting chars) within the given time-frame. Thus, I am convinced that this data is not straight-forwardly identifiable.

The data gathered in this way should give some insights into possible relations between programmers' activity (measured in chars added/deleted), used languages and the number of projects simultaneously worked on.

The user interface (UI), simply called the dashboard, is built with the research participants in mind. The purpose of the visualizations and the controls, that the dashboard provides, is to make the users more comfortable with the tool, to enhance trust in it and to potentially encourage them to use it regularly.

3.5. Flow schema of the tool

The data in the tool flows in several directions. User's action triggers changes in the source files. These changes cause the `TextDocumentChangeEvent` [39] events to be emitted. It is important to note here that the only changes that are tracked internally by VSCode are those associated with the files opened in the editor. Presumably, the reasons behind this mechanism are mainly performance ones. These events are then captured by the internal system. The internal system serves a few purposes (which are described in more detail in the subsection 3.6.1). It is responsible for saving the data in the file system, compressing the data and sending it to the cloud storage (Dropbox in my case), as well as controlling the functions of the dashboard.

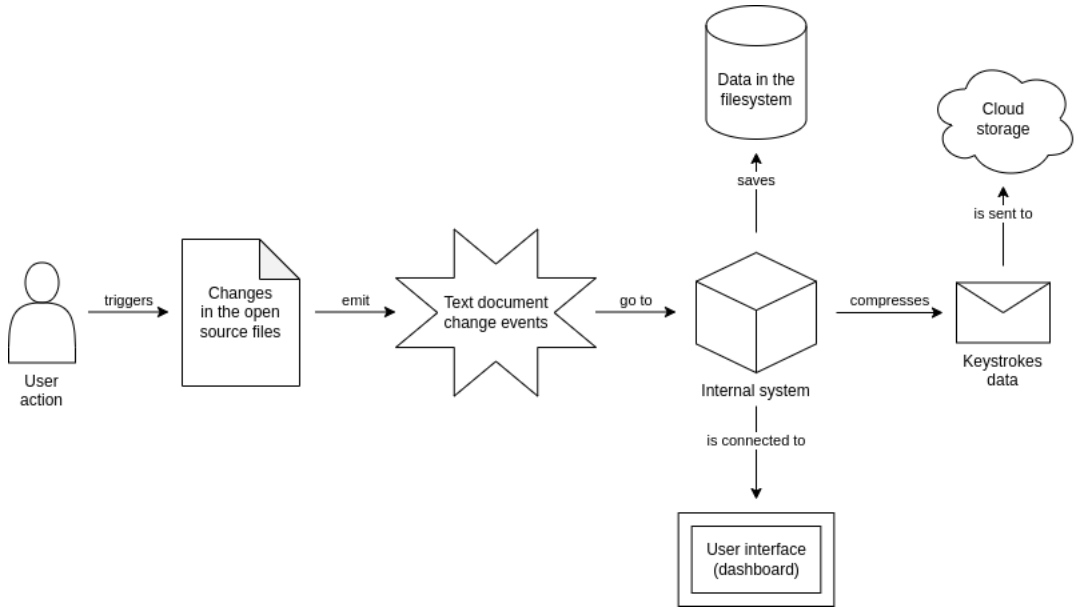

Figure 3.6: Flow schema of the tool

3.6. Crucial elements of the `MIMUW-MB-TT` plugin

3.6.1. The internal system

The main component of the internal system are the two buffers filled with the intervals. One of these buffers is the live buffer. The live buffer contains the data gathered lately. It is responsible for creating new intervals from the data that is coming from the user in the real time. The tool remembers to save the data in the file system in order to preserve it and not

lose it all after potential crashes or closing the tool. After saving the data it is possible to remove some intervals from the live buffer in order to be aware of its size and not to allow it to get too big and clog the memory. The other of the buffers is the static buffer. It is responsible for loading the data from the file system in order for it to be then transferred to the dashboard. The user can request the data from the dashboard.

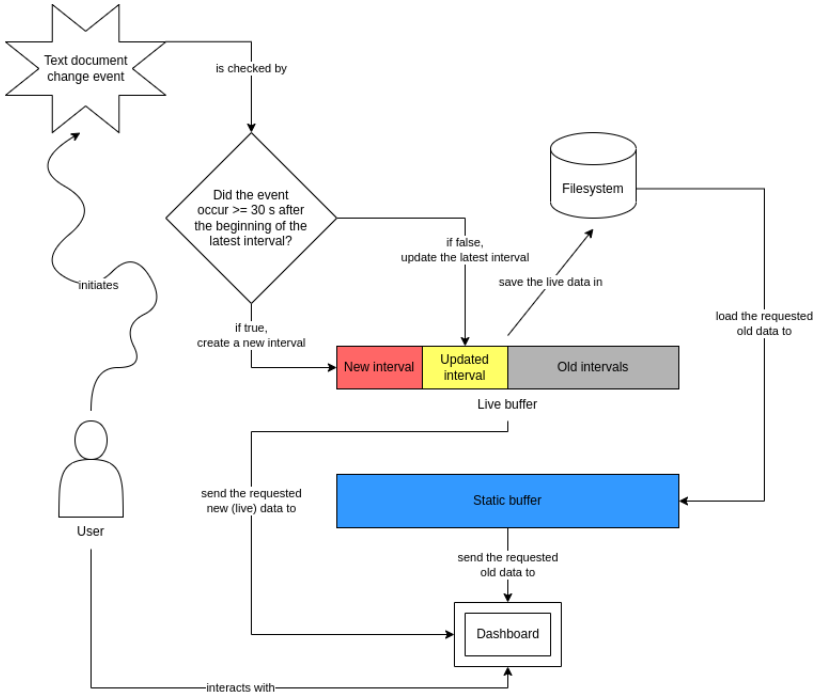

Figure 3.7: Internal system

In order for for the intervals to be visualized they need to be normalized that is made to be of equal lengths. But the raw data consists of the intervals of varying lengths that is of length at most 30 seconds. I need to adjust them to be even-sized. Normalization recognizes also that the intervals need to stick to each other, however in the raw data there may exist breaks, since there are situations where for some time a user may be inactive. So essentially I can treat the time spaces that are not covered by the data as intervals with zero keystrokes. Then by interpreting this data such way I can proceed with interpolation (as shown in the example 3.7). Let me imagine that I split the requested time space into intervals of equal length and then I check if they intersect with the intervals from the real data. The amount of keystrokes that the new interval receives is proportional to the relative size of the intersection to the length of the whole row interval. The total number of keystrokes associated with this new interval is calculated by summing all of the keystrokes associated with all the intersections from the different row intervals.

3.6.2. Dashboard (UI)

The dashboard constitutes the user interface that a user can use to view various statistics across different time periods. It is possible through switching modes. The first mode is called the live view 3.9. In this view the user is able to view the latest data. Hovering over the data points on the plot (green or red dots on the line) reveals a tooltip containing more details (mainly the exact numerical values of the respected data point). The user can also adjust the

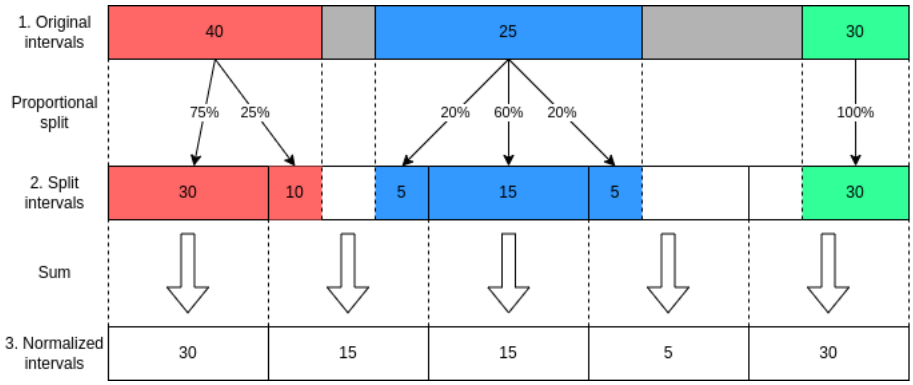

Figure 3.8: An intervals normalization example

time gauge to explore the data in the different time periods (last 10, 30 or 60 minutes from the current time). Moreover, he is provided with the Calendar view 3.10. It is utilized to explore the data from previous days. After selecting the current day the user can also adjust the time period by using the aforementioned gauges. The parameters of the plot are adjusted automatically in order for the plot to be clearly visible to the user. This means to be not too convoluted and not too cluttered. The dashboard is connected with the live and static buffers and depending on the view the data is loaded from the live buffer or from the static buffer (that is indirectly from the file system). It is possible through the day view 3.11 that is accessible by selecting a day in the calendar view.

Users are enabled to save and upload the data manually by clicking on the buttons in the upper-right corner of either of the views. Since the quota of API calls to Dropbox servers is limited with my subscription plan, I have implemented a cooldown period on subsequent manual uploads to conserve the available API invocations.

3.7. Research questions

3.7.1. How to visualize the day of a programmer?

One of the initial goals of my thesis was to present the working day of a programmer in a way that would be straightforward and beneficial for both the programmer and her/his potential employer. I hope that the visualizations created by me 4.1 achieve just that.

3.7.2. What are the activity patters of programmers during the day?

I examine if the programmers of different languages remain active during the day and night. It is achieved by analyzing data points with regards to their timestamps and programming languages used. Additionally, I compute the performance metric measured by the average number of actions per data point to hypothesize about performance fluctuations with regards to moving time. This statistic also takes into account the programming language to search for potential divergence in activity patters between people using different languages.

3.7.3. How do programmers take breaks during the day?

Programmers need breaks between coding to read the code, study documentation, complete various professional tasks or just to have lunch. To analyze these breaks, I compute the lengths of all the recorded spaces between data points within a day (omitting the breaks

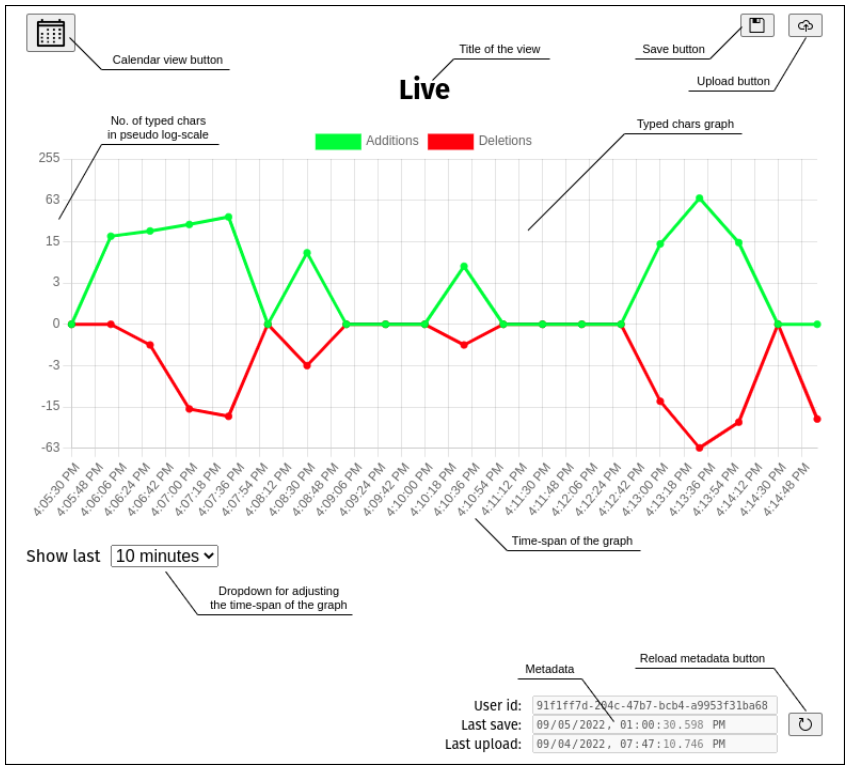

Figure 3.9: Live view of the dashboard

taken between different days). This way it becomes possible to bin the data about breaks and compose a histogram to observe the frequency of shorter and longer breaks for programmers. Again, the programming language is also taken into account here.

3.7.4. What are the proportions of code added to code deleted?

Programmers during coding may not only add the new code, but also delete the old one and replace it. I study the characteristics of this behavior by analyzing the ratio of the number of deletions to the number of additions. Small values of this ratio (below 100%) indicate the increase in the code base, while higher values (over 100%) point to the code being mostly removed. The ratio is calculated separately per data point and per day to find out if the statistical trends hold under diverse granularity levels. Similar check is done on the basis of the difference in the programming language used.

3.7.5. How many projects do programmers work on?

It is quite common for programmers to work on multiple projects in parallel especially since they may constitute the basis of a larger venture. I examine what is the usual number of projects that the programmers work on during the day and overall. Additionally, I check how frequent it is to switch between different projects within one day.

3.7.6. How does the data change in relation to the day of the week?

It is expected that programmers' activity decreases on the weekend. I validate these claims by comparing the number of data points and actions generated in distinct days of the week.

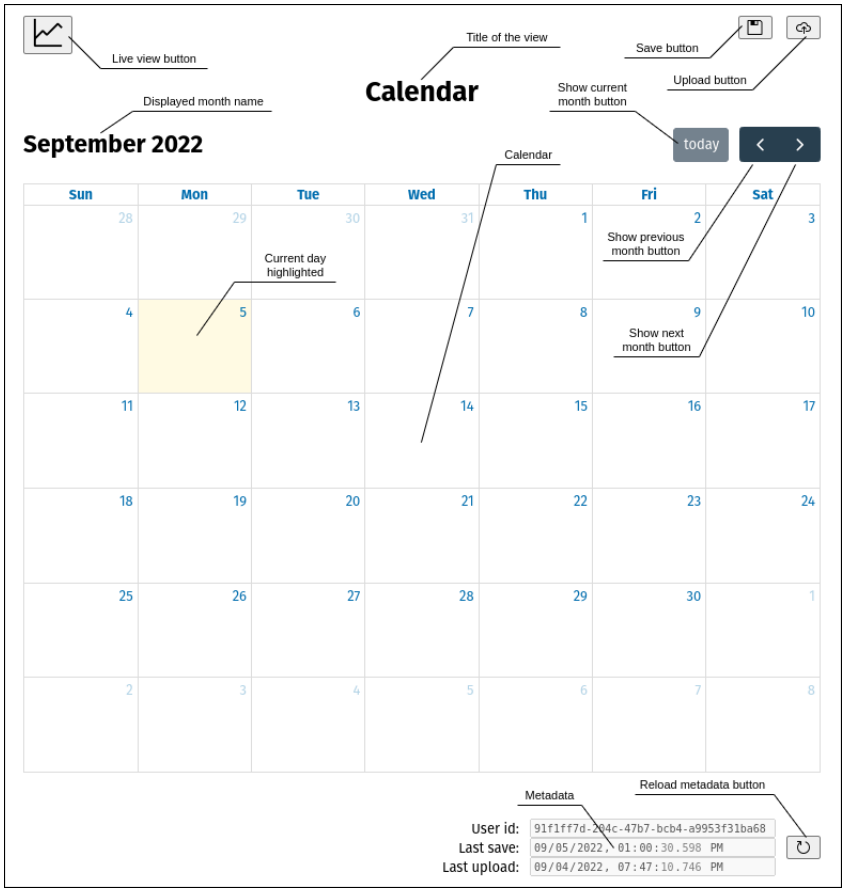

Figure 3.10: Calendar view of the dashboard

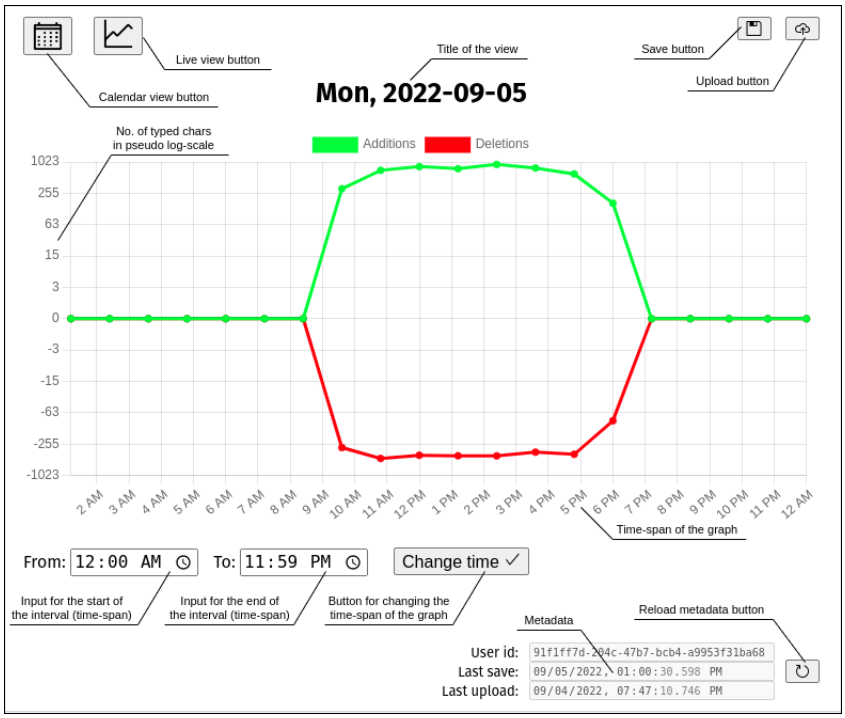

Figure 3.11: Day view of the dashboard

Performance (measured as the average number of actions per data point) is also calculated with this regard to search for potential disparity in programmers' performance between weekdays and the weekend days.

Chapter 4

Results

4.1. Visualization of a day of a programmer

I present two visualizations (as plots): using `additions`/`deletions` 4.1 and `actions` 4.2. Both of them show the values of their respected attributes per unit of time. This way it is easier to consider programmer's features such as his performance. The first visualization makes it possible to analyze the programmer's working day in a more sophisticated way. The numbers of added or deleted characters may indicate the actual tasks done by the developer (i.e. writing, editing or removing parts of the codebase) in the relevant time periods. The second visualization focuses on providing an approximate image of programmer's performance, since `actions` per second can be interpreted as a simple metric of this feature.

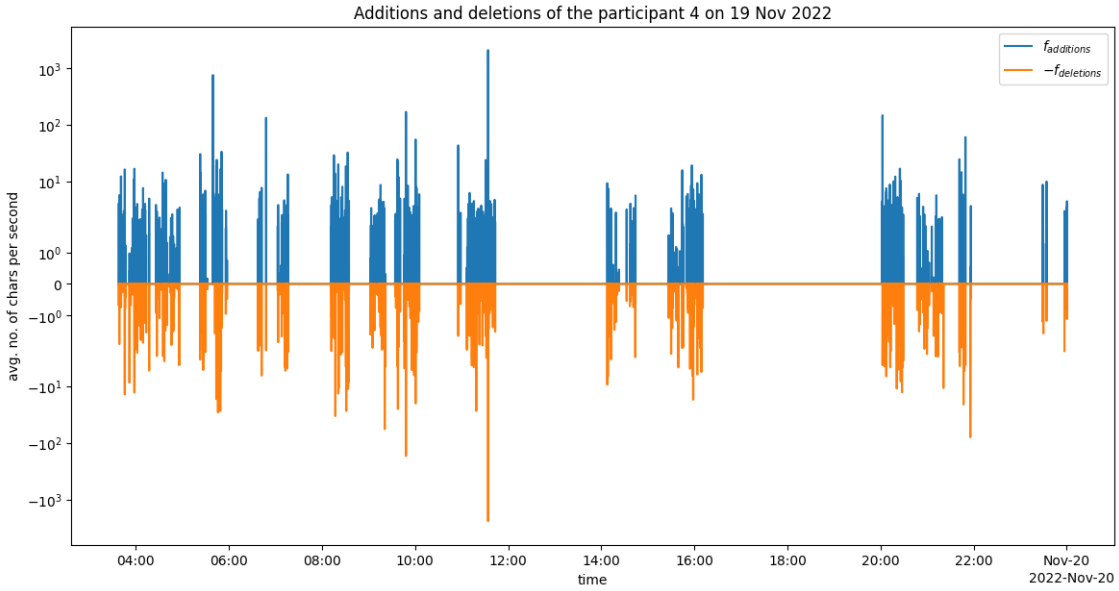

Figure 4.1: Visualization of a programmer's day with `additions` and `deletions`

4.2. Activity

On the figure 4.3, the activity of programmers by hour is presented. It is measured in two ways: by counting the data points generated in distinct hours, and by summing the actions

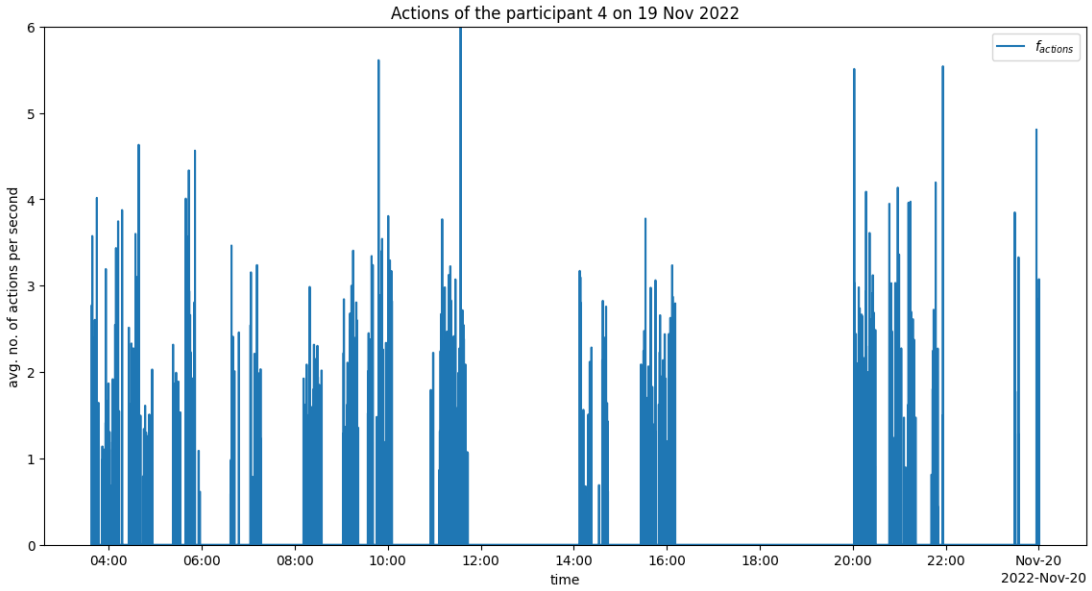

Figure 4.2: Visualization of a programmer's day with `actions`

within these data points. I remind that `actions` is an attribute of a data point 3.3 and here I sum these attributes of multiple data points. In both cases of measurement the `start` timestamp determines the hour bucket where a data point or an action falls under. Timezone from the timestamps is ignored here and in analogous graphs later. Thus, for instance, two programmers working at 10:00 in Warsaw and at 10:00 in San Francisco on the same day would contribute to the same hour bucket. On the graph it is clearly observed in both of the measurement ways that most of the activity happens in the afternoon and that the work almost completely ceases in the middle of the night and around dawn.

Similar observations can be made by examining the activity graph with respect to different programming languages 4.4. The noticeable difference may be that for JS/TS developers the activity is strongly concentrated in the afternoon where for Python developers some activity is also observed during the night.

When it comes to performance measured as the average number of actions per data point, it remains relatively stable 4.5 with visible dips around the end of the standard working day (i.e. between 4 and 6 pm) and at night. Perceived anomalous surges during that period (i.e. at 12 and 5 am) occurred arguably due to smaller samples of data gathered at those time slots.

Such a trend is less clearly visible when restricted to Python and JS/TS languages 4.6, but this may be due to the fact that much less data was gathered at night and in the case of JS/TS even none in the majority of night hours. Comparing the results between the languages, it is noticeable that the performance metric for JS/TS developers is generally higher than for the Python ones (ignoring the night period).

4.3. Breaks

There is no observed striking disparity between programmers of different languages and overall 4.7. Understandably, the longest breaks are much less frequent than the shorter ones. Nevertheless, it is not the shortest breaks that are the most frequent, but the middle length

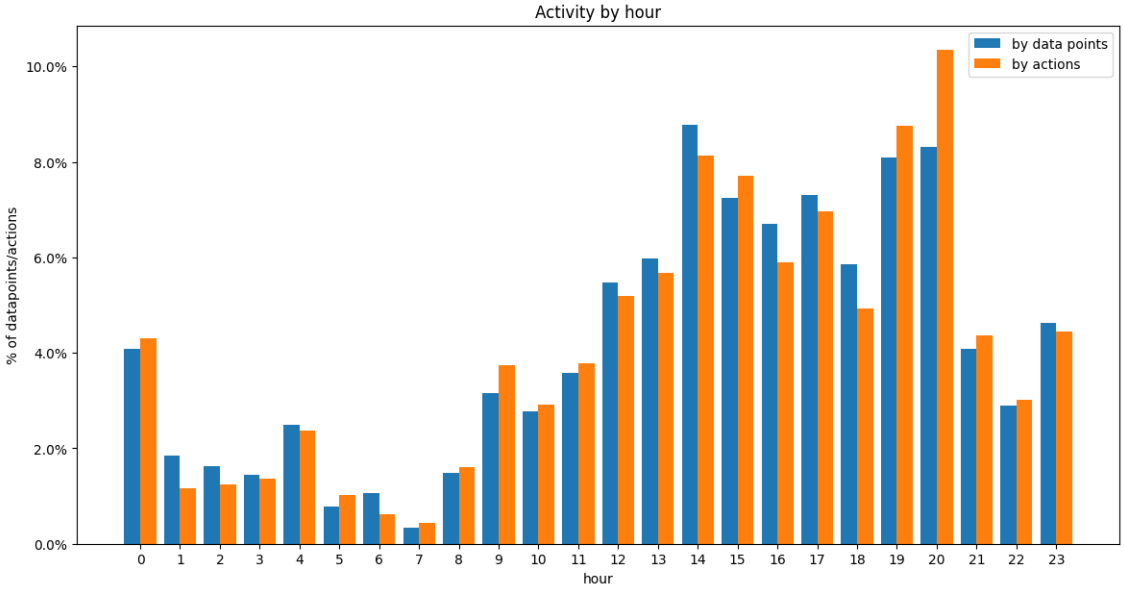

Figure 4.3: Comparison of activity by hour with regards to data points and actions

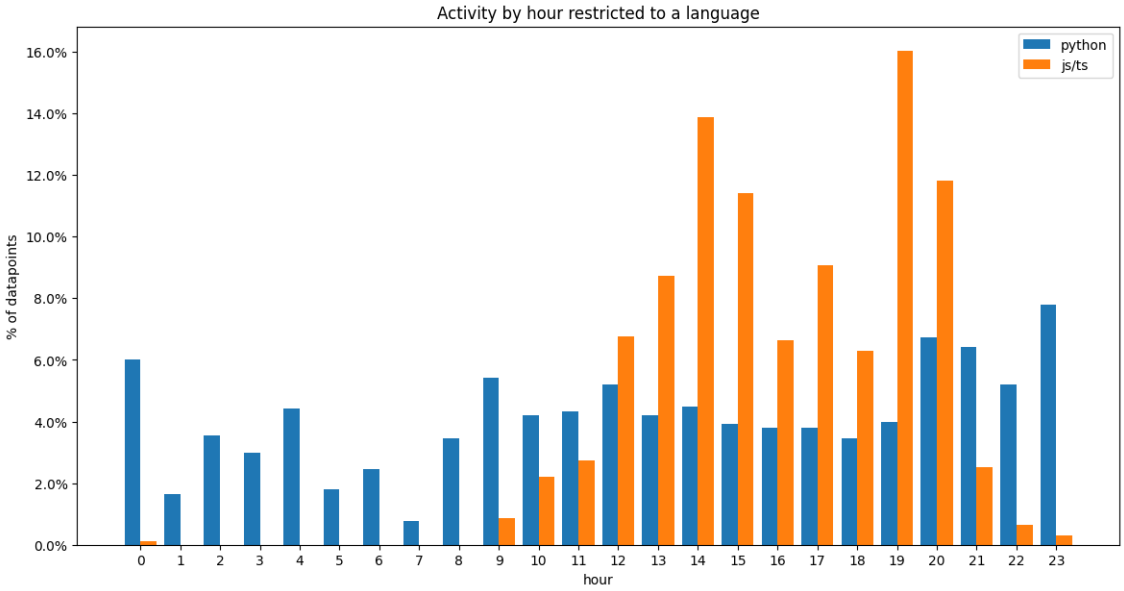

Figure 4.4: Comparison of activity by hour restricted to a language

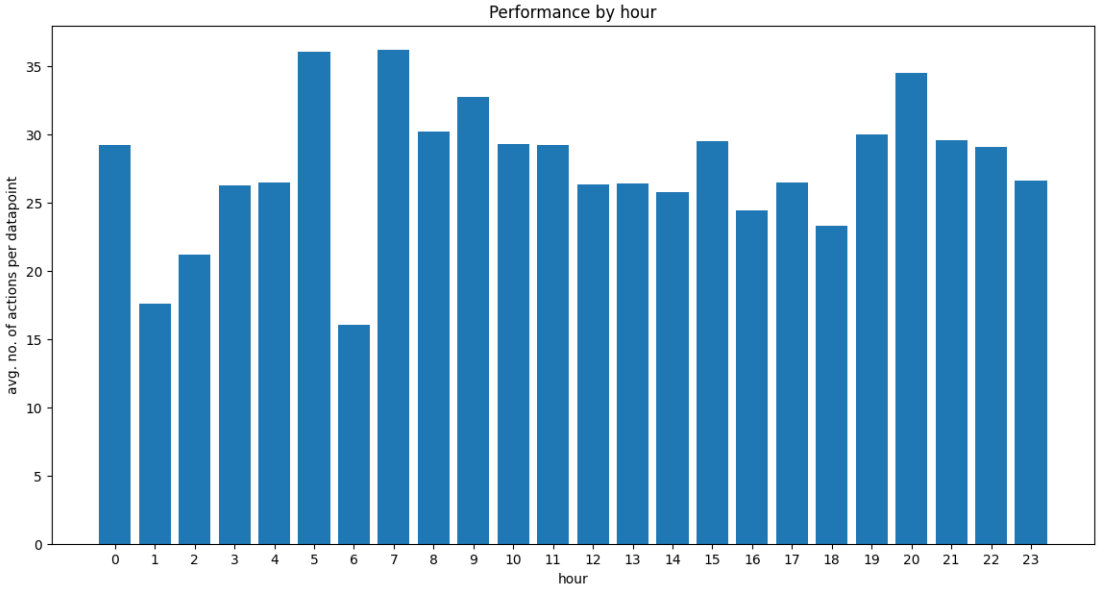

Figure 4.5: Performance by hour

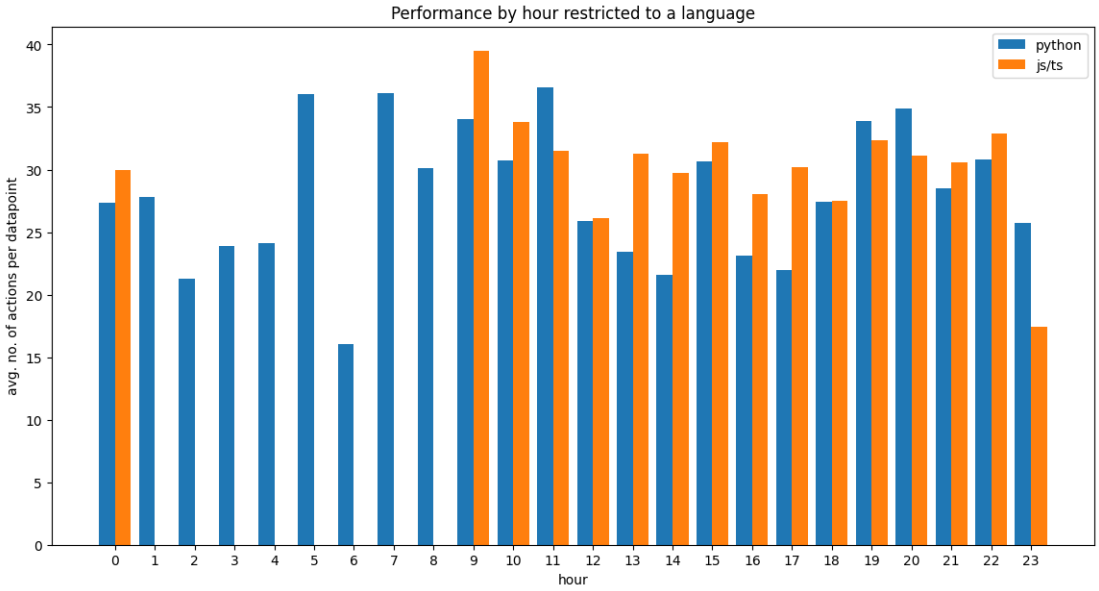

Figure 4.6: Performance by hour restricted to a language

breaks of about 30 seconds.

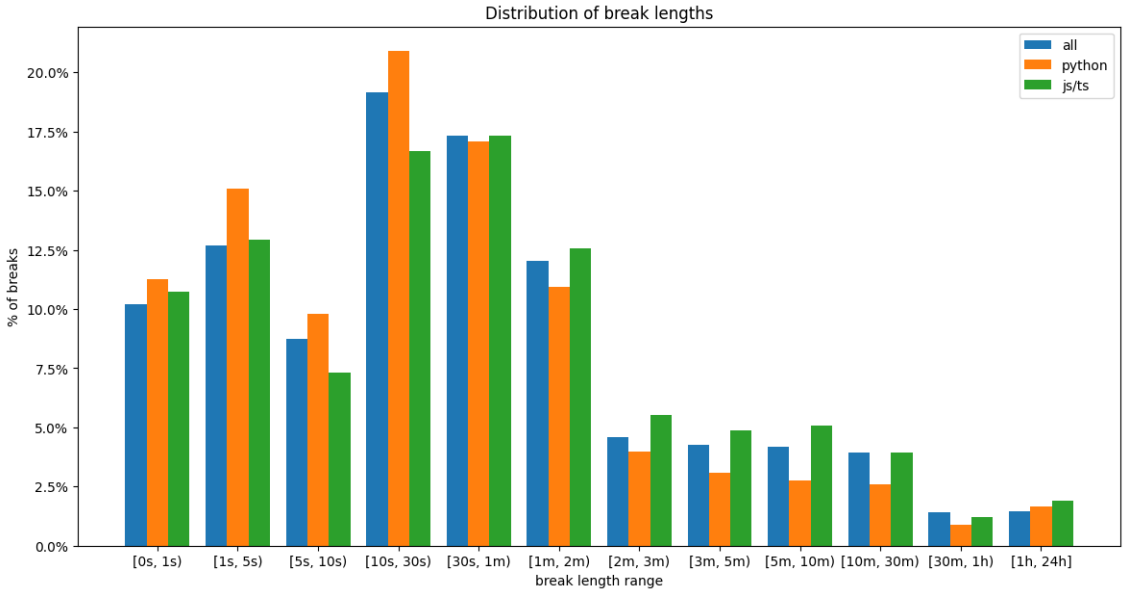

Figure 4.7: Distribution of break lengths overall and restricted to a language

4.4. Deletions to additions

For both granularities (per data point and per day), there is a trend of decreasing frequency of ratios with the increase of deletions relative to additions 4.8. There are two apparent exceptions from this tendency. The days with a slightly higher number of deletions to additions are much more frequent than in the case of data points. Apart from that, quite common are occurrences where the number of deletions is much higher than additions (signified by the last bars on the right of the plots).

There are no particular differences between Python and JS/TS developers with regards to the regarded ratio. One thing that is visible is the higher occurrence of the data points with lower ratio (i.e. more additions) in the case of Python comparing to JS/TS.

4.5. The number of projects

Counting projects is possible thanks to `workspaceNameHash` 3.3. In the vast majority of cases a programmer works on one project per day according to the data 4.10. Cases with 5 or more projects were not recorded. Situations where no projects were recorded during the day (i.e. a user in VS Code works outside the named workspace) remain very scarce.

A similar picture is observed even when considering all the projects of one person within the time frame of my research 4.11. Cases with 6 or more recorded projects per person were uncommon and most oscillated between 1 and 3.

It turns out that according to the data it is also uncommon to switch projects during the working day 4.12. During the staggering most of the days developers did not switch the project even once.

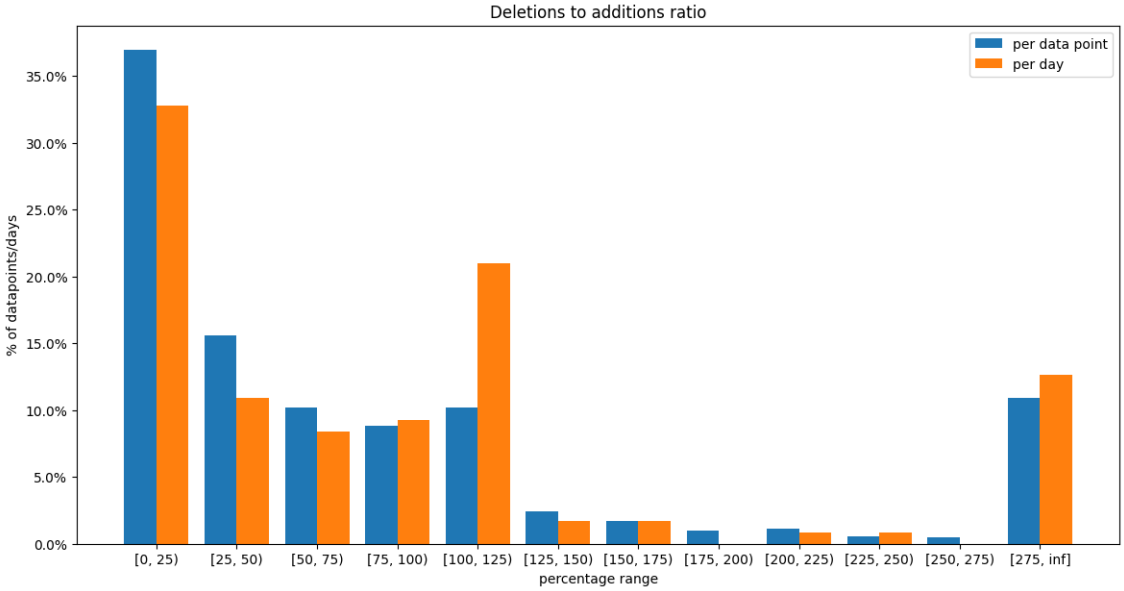

Figure 4.8: Deletions to additions with regards to data points and days

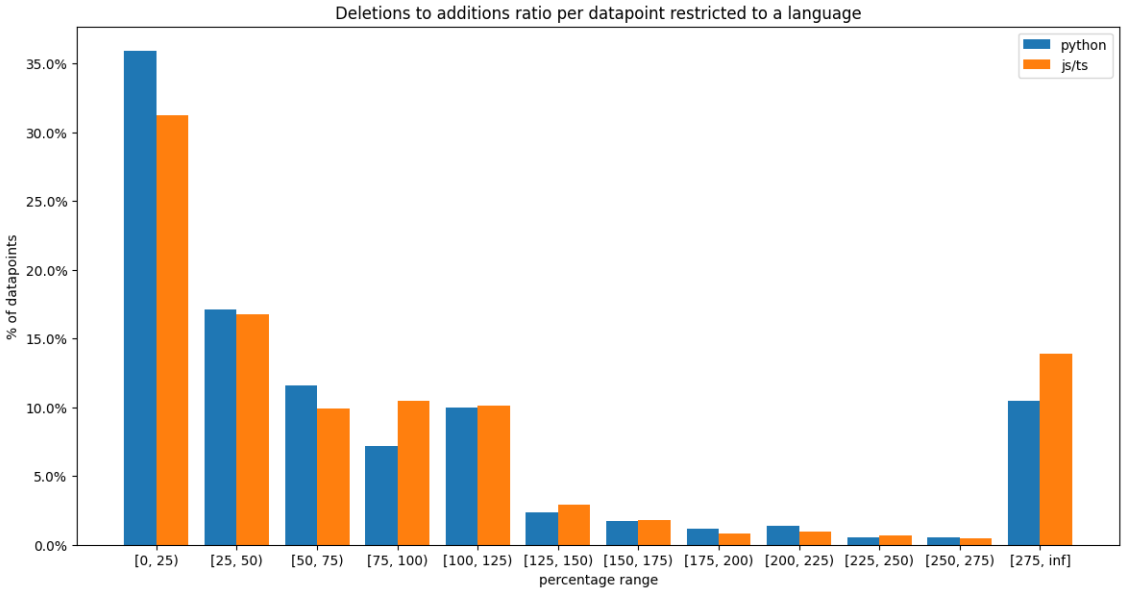

Figure 4.9: Deletions to additions restricted to a language

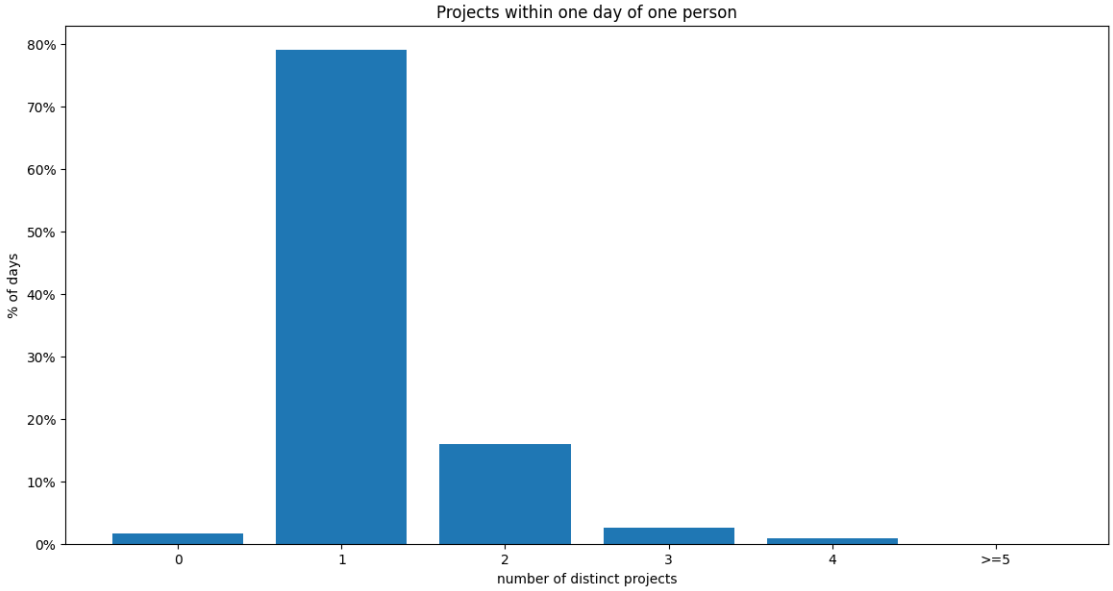

Figure 4.10: Number of projects within one day of a programmer

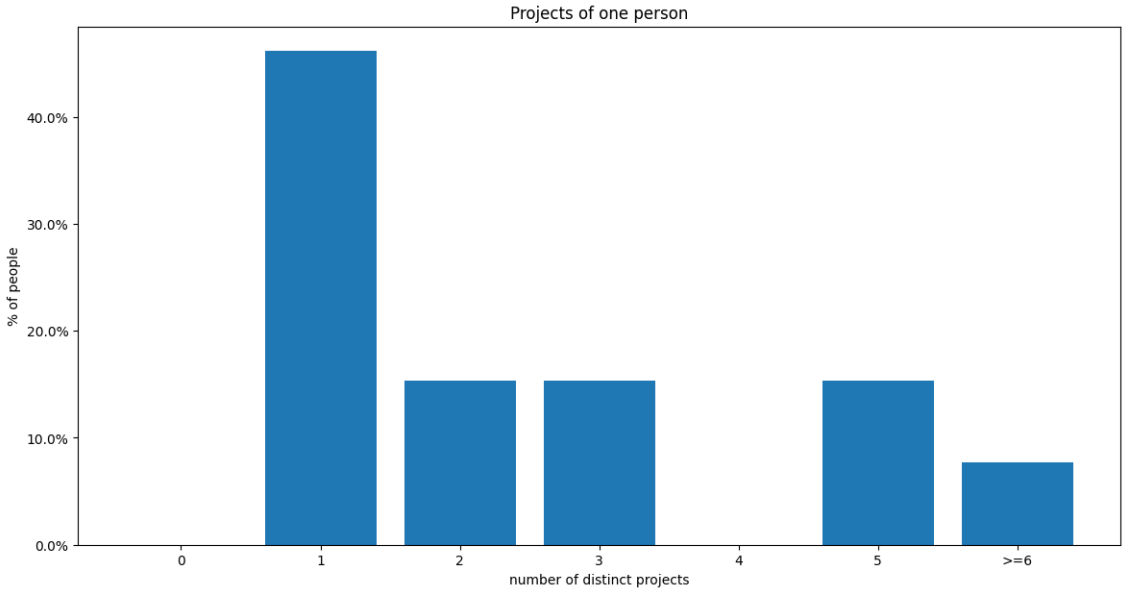

Figure 4.11: Number of programmer's projects

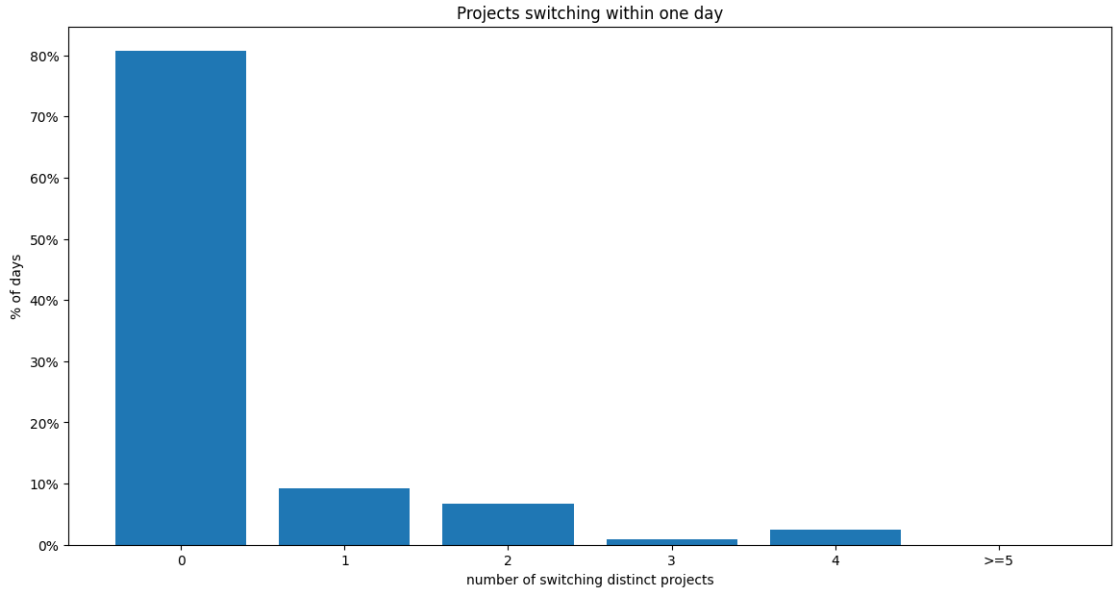

Figure 4.12: Number of projects switching within one day of a programmer

4.6. Per weekday differences

The activity peaks in the middle of the week and decreases on Friday, only to slump during the weekend 4.13. Regardless of the measurement method (by counting data points or summing up actions), the aforementioned trends seem consistent.

Analysis of performance did not give any conclusive results with them appearing relatively stable 4.14.

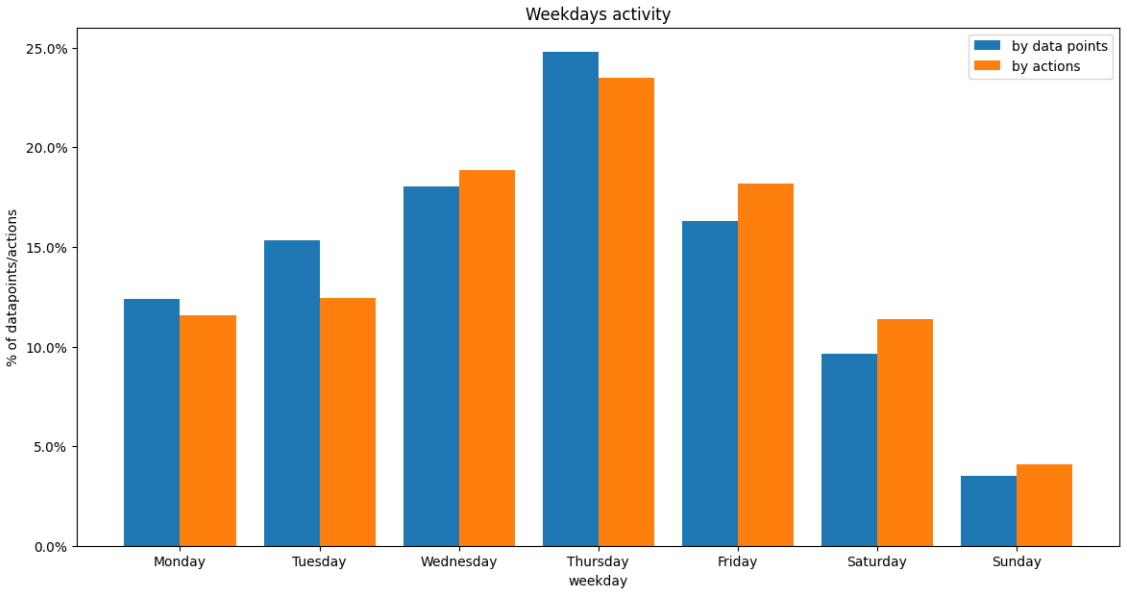

Figure 4.13: Activity per weekday with regards to data points and actions

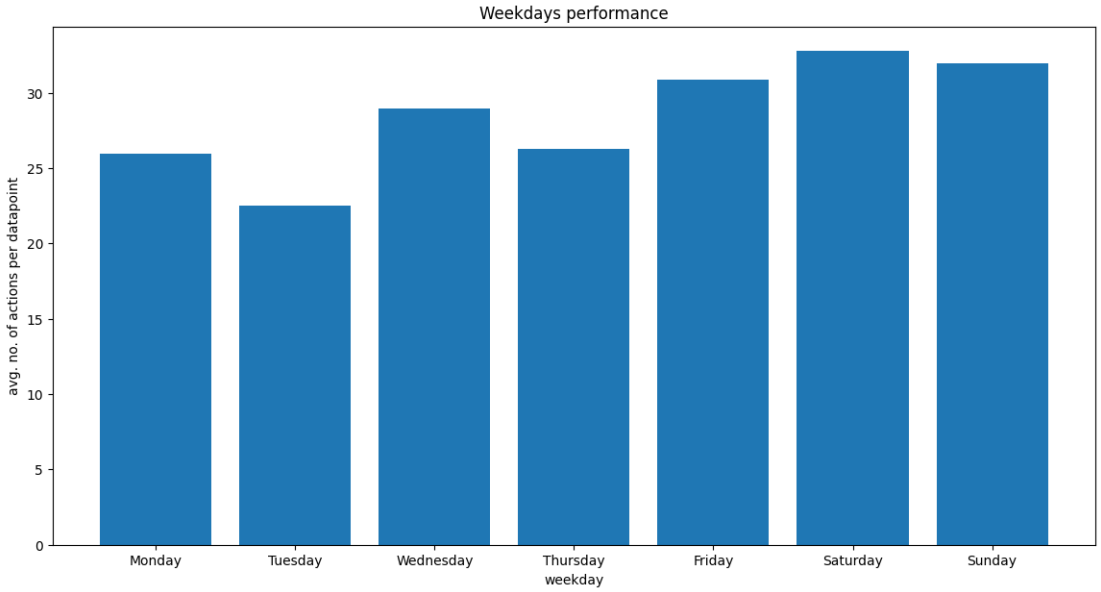

Figure 4.14: Performance per weekday

Chapter 5

Discussion

5.1. Revisiting results

Analysis of the data gathered by the `MIMUW-MB-TT` plugin sheds some light onto the characteristics of programmers' working day.

Visualizations of a working day of a programmer give potentially valuable insights into the programming process. The developer can introspect and measure his performance. His employer is able to more accurately assess the productivity and potentially provide the assistance where it is needed the most.

Activity analysis shows that understandably most of the work concentrates in the afternoon, but with a notable portion of it done in the evening. Similar patterns emerge when analysing the data from programmers of Python and JS/TS separately. Nevertheless, no sound evidence towards difference of chronotypes between programmers of different languages were discovered.

Breaks analysis unsurprisingly concludes that longer breaks are less frequent that shorter ones and this observation holds for different programming languages and overall. The reason why middle length breaks are more frequent than the shortest ones may just be related to the method of measurement with a break considered as a time space between two data points. Let me remind here that one data point holds information of at most 30 seconds.

Analysis of the ratio of deletions to additions shows that it is more common to add code than to delete it and this trend stays true regardless of the programming language used.

The number of projects per person is observed as at most just a few. Moreover, frequent switching between different projects was not observed.

The data supports the common knowledge that activity dips on weekends. Additionally, it decreases even earlier on Friday which was observed also by Robinson [20].

5.2. Threats to validity

5.2.1. Doubts about data gathered

I gathered the data from 15 programmers. This number comes from counting the data packages I have received throughout the study. I assume that each of them comes from a different person, although it is theoretically possible that one person uses the `MIMUW-MB-TT` plugin on separate workstations, thus is counted multiple times. From my personal talks with the participants, I assess the probability of such occurrences as low.

Although the participants were instructed to use the plugin for at least two weeks, the

number of downloads significantly surpass the number of data packages (a minority of them were test downloads by me) and even within the packages there are differences of size between them. Because of that, some participants may influence the final results stronger than the rest.

5.2.2. Questions about participants in the study

Apart from the number of participants, it is important to remember they were not chosen uniformly from the whole community of programmers. Most of them came from the MIMUW faculty as student and it may significantly incluence the final results.

5.2.3. `MIMUW-MB-TT` plugin shortcomings

Due to the technical obstacles, the plugin does not save the data in the file system right before closing the VS Code window. It just saves the data on 2 minutes interval, so it is possible to lose the data gathered right before closing the IDE.

Additionally, there exist an exploit related to project switching. Since `workspaceNameHash` relates to the last workspace name recorded for a data point, it is theoretically possible to work within 30 second period on different projects and the system would store such an event as working on just one. This would, however, involve quite a deep knowledge of the plugin and rather unnatural way of typing, so I deem the risk of this exploit influencing the gathered data as minimal.

Apart from that, the plugin allows a situation where `start` and `end` are equal for a data point. While this does not constitute a bug by itself, it creates potential confusion and inconvenience when analyzing the gathered data.

Chapter 6

Conclusions and future work

6.1. Answers to research questions

Let me sum up the conclusions according to the gathered data.

I suggested 2 visualizations of programmer's working day. Both of them have the potential to assist developers with their growth and their employers with better employee evaluation and resource management. This may be especially pertinent in the present IT job market [51].

Programmers work mostly during standard work hours with smaller portions of it done during the late evening. Differences in chronotypes between programmers of Python and JS/TS remain inconclusive.

Shorter breaks are much more frequent than longer ones, however the ones of length around 30 seconds are the most frequent and not the shortest ones. This may be due to the way how the plugin is constructed or really there is something tangible here about programmers' behavior.

Analysis of the ratio of deletions to additions show that programmers on average more often add code than delete it. It may be an indicator of the further growth of existing codebases or the constant creation of new ones.

During the span of the study, participants were observed to work only on a handful of distinct projects. This observation is especially relevant during the current trend of overemployment [42] that is working multiple jobs concurrently at the same time [41].

Differences in activity per days of the week are most visible when comparing normal weekdays with the weekend when the activity significantly drops. Performance calculations did not give conclusive insights into potential differences in performance between different days of the week.

6.2. Key contributions

- Simple visualizations of programmer's working day.
- Published open-source plugin to gather the programmers' data.
- Published open-source code to analyze the gathered programmers' data.
- Insights into the programming process of programmers regarding multiple variables.

6.3. Future work

The analysis and visualization of programming process is an extensive field of study. Because of that, it is possible to extend my study in many ways. Below I present my suggestions.

Gathering more data is an obvious choice for future work. That would make the results more believable and non-attributable as just some sort of a fluke. It would be important to check the programmers background to prepare a statistically appropriate group of study participants. To accomplish this, a more sophisticated rewarding scheme may be needed e.g. like the one from the DARPA Red Balloon Challenge [43].

Extending the capabilities of the `MIMUW-MB-TT` plugin also seems like a straight-forward idea. Labeling the events as concrete programmers' actions (like switching a git branch) would require much more involvement from the participants (by requiring them to annotate the data points), but again would contribute to making the study more serious. Publishing the gathered data is also a necessary step in this direction that may be achieved by implementing privacy-preserving data collection [45]. All these improvements could potentially give the `MIMUW-MB-TT` plugin a chance to be as influential as some similar tools integrated with IDEs other than VS Code [44, 46, 47].

Multiple hypotheses may be constructed from my thesis. One of them that intrigues me is that in the figure 4.8 the presented ratios of deletions to additions seem to potentially be invariant to differences in data granularity. It may be interesting to challenge this claim.

Chapter 7

Data and code availability

In order to streamline my processes and make them more appealing to the public and to mitigate the risk of de-anonymization [40], a part of the agreement was not to publish any keystroke data of my testing subjects. At GitHub I have published the source code of my plugin, the source code available for data analysis and the Latex source code of this thesis. Anyone also can download my plugin directly from VS Code or at Visual Studio Marketplace 7.1.

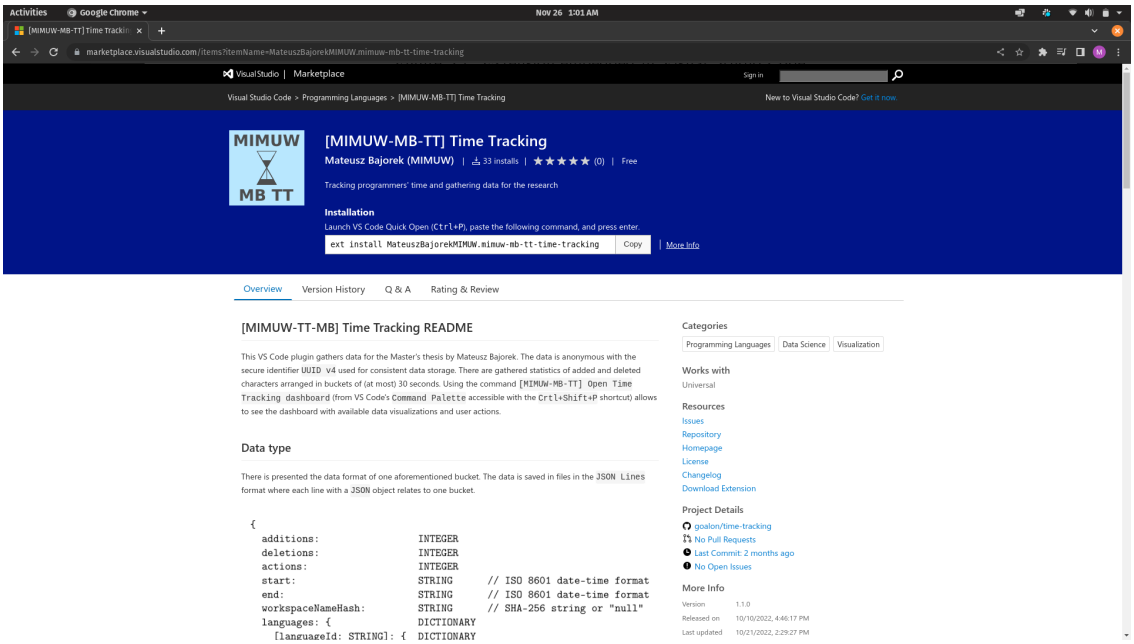

Figure 7.1: *MIMUW-MB-TT* plugin in the VS Marketplace (as of 26 Nov 2022)

Bibliography

[1] Steven M. Reppert, and David R. Weaver. 2002. *Coordination of circadian timing in mammals*. Nature 418, 935–941, doi: 10.1038/nature00965.

[2] Till Roenneberg, Anna Wirz-Justice, and Martha Merrow. 2003. *Life between Clocks: Daily Temporal Patterns of Human Chronotypes*. Journal of Biological Rhythms, 18(1), pp. 80–90, doi: 10.1177/0748730402239679.

[3] Dale Stokdyk. 2021. *What Do Programmers Do, Anyway?* Southern New Hampshire University, url: `https://www.snhu.edu/about-us/newsroom/stem/what-do-programmers-do`.

[4] Tony Fedorenko. 2021. *Time Tracking in Software Development*. Mad Devs, url: `https://maddevs.io/customer-university/time-tracking-in-software-development/`.

[5] Growin. 2021. *A Brief History of Software Development Methodologies*. Growin Blog, url: `https://www.growin.com/blog/history-of-software-development-methodologies/`.

[6] Richard C. Thomas, Amela Karahasanovic, and Gregor E. Kennedy. 2005. *An investigation into keystroke latency metrics as an indicator of programming performance*. Proceedings of the 7th Australasian conference on Computing education - Volume 42 (ACE '05). Australian Computer Society, Inc., AUS, pp. 127–134, doi: 10.5555/1082424.1082440.

[7] Juho Leinonen, Krista Longi, Arto Klami, and Arto Vihavainen. 2016. *Automatic Inference of Programming Performance and Experience from Typing Patterns*. Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16). Association for Computing Machinery, New York, NY, USA, pp. 132–137, doi: 10.1145/2839509.2844612.

[8] Microsoft. 2022. *Compiling TypeScript*. Visual Studio Code Documentation, Microsoft, url: `https://code.visualstudio.com/docs/typescript/typescript-compiling`.

[9] Juho Leinonen. 2019. *Keystroke Data in Programming Courses*. Ph.D. Dissertation, University of Helsinki.

[10] Microsoft. 2022. *Programming Languages / Changing the language for the selected file*. Visual Studio Code Documentation, Microsoft, url: `https://code.visualstudio.com/docs/languages/overview#_changing-the-language-for-the-selected-file`.

[11] Albina Zavgorodniaia, Raj Shrestha, Juho Leinonen, Arto Hellas, and John Edwards. *Morning or Evening? An Examination of Circadian Rhythms of CS1 Students*. 2021

IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET), 2021, pp. 261-272, doi: 10.1109/ICSE-SEET52601.2021.00036.

[12] Dror G. Feitelson, Ayelet Mizrahi, Nofar Noy, Aviad Ben Shabat, Or Eliyahu, and Roy Sheffer. *How Developers Choose Names*. IEEE Transactions on Software Engineering, vol. 48, no. 1, pp. 37-52, 1 Jan. 2022, doi: 10.1109/TSE.2020.2976920.

[13] Yoshiaki Matsuzawa, Ken Okada, and Sanshiro Sakai. 2013. *Programming process visualizer: a proposal of the tool for students to observe their programming process*. Proceedings of the 18th ACM conference on Innovation and technology in computer science education (ITiCSE '13). Association for Computing Machinery, New York, NY, USA, pp. 46–51, doi: 10.1145/2462476.2462493.

[14] Jacob T. Biehl, Mary Czerwinski, Greg Smith, and George G. Robertson. 2007. *FASTDash: a visual dashboard for fostering awareness in software teams*. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07). Association for Computing Machinery, New York, NY, USA, pp. 1313–1322, doi: 10.1145/1240624.1240823.

[15] Elena Lyulina, Anastasiia Birillo, Vladimir Kovalenko, and Timofey Bryksin. 2021. *TaskTracker-tool: A Toolkit for Tracking of Code Snapshots and Activity Data During Solution of Programming Tasks*. Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21). Association for Computing Machinery, New York, NY, USA, pp. 495–501, doi: 10.1145/3408877.3432534.

[16] Raj Shrestha, Juho Leinonen, Arto Hellas, Petri Ihantola, and John Edwards. 2022. *CodeProcess Charts: Visualizing the Process of Writing Code*. Australasian Computing Education Conference. Association for Computing Machinery, New York, NY, USA, pp. 46–55, doi: 10.1145/3511861.3511867.

[17] Evan Balzuweit, and Jaime Spacco. 2013. *SnapViz: visualizing programming assignment snapshots*. Proceedings of the 18th ACM conference on Innovation and technology in computer science education (ITiCSE '13). Association for Computing Machinery, New York, NY, USA, pp. 350, doi: 10.1145/2462476.2465615.

[18] Cindy Norris, Frank Barry, James B. Fenwick Jr., Kathryn Reid, and Josh Rountree. 2008. *ClockIt: collecting quantitative data on how beginning software developers really work*. Proceedings of the 13th annual conference on Innovation and technology in computer science education (ITiCSE '08). Association for Computing Machinery, New York, NY, USA, pp. 37–41, doi: 10.1145/1384271.1384284.

[19] Maëlick Claes, Mika V. Mäntylä, Miikka Kuutila, and Bram Adams. 2018. *Do programmers work at night or during the weekend?* Proceedings of the 40th International Conference on Software Engineering (ICSE '18). Association for Computing Machinery, New York, NY, USA, pp. 705–715, doi: 10.1145/3180155.3180193.

[20] David Robinson. 2017. *What Programming Languages Are Used Late at Night?* Stack Overflow Blog, url: `https://stackoverflow.blog/2017/04/19/programming-languages-used-late-night/`.

[21] Julia Silge. 2017. *What Programming Languages Are Used Most on Weekends?* Stack Overflow Blog, url: `https://stackoverflow.blog/2017/02/07/what-programming-languages-weekends/`.

[22] Renato Lima Novais, André Torres, Thiago Souto Mendes, Manoel Mendonça, and Nico Zazworka. 2013. *Software evolution visualization: A systematic mapping study.* Information and Software Technology, Volume 55, Issue 11, 2013, pp. 1860-1883, ISSN 0950-5849, doi: 10.1016/j.infsof.2013.05.008.

[23] Anna-Liisa Mattila, Petri Ihantola, Terhi Kilamo, Antti Luoto, Mikko Nurminen, and Heli Väätäjä. 2016. *Software visualization today: systematic literature review.* Proceedings of the 20th International Academic Mindtrek Conference (AcademicMindtrek '16). Association for Computing Machinery, New York, NY, USA, pp. 262–271, doi: 10.1145/2994310.2994327.

[24] Arto Vihavainen, Matti Luukkainen, and Petri Ihantola. 2014. *Analysis of source code snapshot granularity levels.* Proceedings of the 15th Annual Conference on Information technology education (SIGITE '14). Association for Computing Machinery, New York, NY, USA, pp. 21–26, doi: 10.1145/2656450.2656473.

[25] Matthew C. Jadud. 2006. *Methods and tools for exploring novice compilation behaviour.* Proceedings of the second international workshop on Computing education research (ICER '06). Association for Computing Machinery, New York, NY, USA, pp. 73–84, doi: 10.1145/1151588.1151600.

[26] Watts S. Humphrey. 1996. *Using a defined and measured Personal Software Process.* IEEE Software, vol. 13, no. 3, pp. 77-88, May 1996, doi: 10.1109/52.493023.

[27] Stephan Diehl. 2007. *Software visualization: visualizing the structure, behaviour, and evolution of software.* Springer Science & Business Media, isbn: 978-3-540-46505-8, doi: 10.1007/978-3-540-46505-8.

[28] Danny T. Connors. 1992. *Software development methodologies and traditional and modern information systems.* SIGSOFT Softw. Eng. Notes 17, 2 (April 1992), pp. 43–49, doi: 10.1145/130840.130843.

[29] Arcady A. Putilov, Nele Marcoen, Daniel Neu, Nathalie Pattyn, and Olivier Mairesse. 2019. *There is more to chronotypes than evening and morning types: Results of a large-scale community survey provide evidence for high prevalence of two further types.* Personality and Individual Differences, Volume 148, 2019, pp. 77-84, ISSN 0191-8869, doi: 10.1016/j.paid.2019.05.017.

[30] Stack Exchange Inc. 2022. *2022 Developer Survey.* Stack Overflow, url: `https://survey.stackoverflow.co/2022/`.

[31] Andrew J. Cohen, Sam Washington, Christi Butler, Puneet Kamal, German Patino, Anas Tresh, Jorge Mena, Medina Ndoye, and Benjamin N. Breyer. 2019. *Altruistic donation to improve survey responses: a global randomized trial.* BMC Res Notes, 28 Feb 2019, 12(1):113, doi: 10.1186/s13104-019-4146-y.

[32] Elisabeth Brüggen, Martin Wetzels, Ko De Ruyter, and Niels Schillewaert. 2011. *Individual Differences in Motivation to Participate in Online Panels: The Effect on Response Rate and Response Quality Perceptions.* International Journal of Market Research, 2011, pp. 369-390, 53, 3, doi: 10.2501/IJMR-53-3-369-390.

[33] U.S. Department of Commerce and National Institute of Standards and Technology. 2012. *Secure Hash Standard - SHS: Federal Information Processing Standards Publication 180-4*. CreateSpace Independent Publishing Platform, North Charleston, SC, USA, isbn: 978-1-4781-7807-1, doi: book/10.5555/2408113.

[34] Juho Leinonen, Petri Ihantola, and Arto Hellas. 2017. *Preventing Keystroke Based Identification in Open Data Sets*. Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale (L@S '17), Association for Computing Machinery, New York, NY, USA, pp. 101–109, doi: 10.1145/3051457.3051458.

[35] Karla D'Alleva Valas. 2022. *3 Trends Shaping Philanthropy In 2022*. Forbes, url: `https://www.forbes.com/sites/karladallevavalas/2022/03/31/3-trends-shaping-philanthropy-in-2022/`.

[36] Adam Rhodes. 2022. *When should you use Files and Documents?*. OpenAI, url: `https://help.openai.com/en/articles/5423507-when-should-you-use-files-and-documents`.

[37] Ian Ward. 2022. *JSON Lines Documentation Main Page*. JSON Lines, url: `https://jsonlines.org/`.

[38] Linux Foundation. 2006. *EULA Definition*. The Linux Information Project (LINFO), url: `http://www.linfo.org/eula.html`.

[39] Microsoft. 2022. *VS Code API*. Visual Studio Code, url: `https://code.visualstudio.com/api/references/vscode-api`.

[40] Arvind Narayanan, and Vitaly Shmatikov. 2008. *Robust De-anonymization of Large Sparse Datasets*. Proceedings of the 2008 IEEE Symposium on Security and Privacy (SP '08). IEEE Computer Society, USA, pp. 111–125, doi: 10.1109/SP.2008.33.

[41] Julia Hiemer. 2022. *Between desirability and reality : conceptualization, measurement, causes, and consequences of overemployment*. University of Bamberg Press, doi: 10.20378/irb-53565.

[42] Bryan Lufkin. 2021. *The 'overemployed' workers juggling remote jobs*. BBC, url: `https://www.bbc.com/worklife/article/20210927-the-overemployed-workers-juggling-remote-jobs`.

[43] John C. Tang, Manuel Cebrian, Nicklaus A. Giacobe, Hyun-Woo Kim, Taemie Kim, and Douglas "Beaker" Wickert. 2011. *Reflecting on the DARPA Red Balloon Challenge*. Communications of the ACM, 54, 4 (April 2011), pp. 78–85, doi: 10.1145/1924421.1924441.

[44] Sven Amann, Sebastian Proksch, and Sarah Nadi. 2016. *FeedBaG: An interaction tracker for Visual Studio*. IEEE 24th International Conference on Program Comprehension (ICPC), 2016, pp. 1-3, doi: 10.1109/ICPC.2016.7503741.

[45] Yue Cao, Liang Wang, Zhiwen Zheng, and Xianping Tao. 2020. *A Tool for Non-Intrusive and Privacy-Preserving Developers' Programming Activity Data Collection*. IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC), 2020, pp. 953-962, doi: 10.1109/COMPSAC48688.2020.0-147.

[46] YoungSeok Yoon, and Brad A. Myers. 2011. *Capturing and analyzing low-level events from the code editor*. Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools (PLATEAU '11). Association for Computing Machinery, New York, NY, USA, pp. 25–30, doi: 10.1145/2089155.2089163.

[47] Roberto Minelli, and Michele Lanza. 2013. *Visualizing the workflow of developers*. 2013 First IEEE Working Conference on Software Visualization (VISSOFT), 2013, pp. 1-4, doi: 10.1109/VISSOFT.2013.6650531.

[48] Roberto Minelli, Andrea Mocci, Michele Lanza, and Lorenzo Baracchi. 2014. *Visualizing Developer Interactions*. Proceedings of the 2014 Second IEEE Working Conference on Software Visualization (VISSOFT '14). IEEE Computer Society, USA, pp. 147–156, doi: 10.1109/VISSOFT.2014.31.

[49] Gabriele Di Rosa, Andrea Mocci, and Marco D'Ambros. 2020. *Visualizing Interaction Data Inside & Outside the IDE to Characterize Developer Productivity*. Working Conference on Software Visualization (VISSOFT), 2020, pp. 38-48, doi: 10.1109/VISSOFT51673.2020.00009.

[50] Oleksandr Zhevaho. 2021. *An Overview of Tools for Collecting Data on Software Development and Debugging Processes from Integrated Development Environments*. Science and Transport Progress Bulletin of Dnipropetrovsk National University of Railway Transport, 3, pp. 24-37, doi: 10.15802/stp2021/242042.

[51] Noor Nanji. 2022. *Elon Musk tells Twitter staff to work long hours or leave*. BBC News, url: `https://www.bbc.com/news/business-63648505`.