Supporting Software Developers Through a Gaze-Based Adaptive IDE

Thomas Weber
LMU Munich
Munich, Germany
thomas.weber@ifi.lmu.de

Rafael Vinicius Mourão Thiel
LMU Munich
Munich, Germany
r.thiel@campus.lmu.de

Sven Mayer
LMU Munich
Munich, Germany
info@sven-mayer.com

Figure 1: Our gaze-based adaptive IDE extension changes the layout of the IDE depending on the developers' intent. For example, when the prediction models determine an increased relevance for the terminal output in the IDE (1), the code editor (2) and other UI elements (3) shrink while the output panel (1) grows in size. This can highlight or reveal otherwise missed information.

ABSTRACT

Highly complex systems, such as software development tools, constantly gain features and, consequently, complexity and, thus, risk overwhelming or distracting the user. We argue that automation and adaptation could help users to focus on their work. However, the challenge is to correctly and promptly determine when to adapt what, as often the users' intent is unclear. To assist software developers, we build a gaze-adaptive integrated development environment using the developers' gaze as the source for learning appropriate adaptation. Beyond our experience of using gaze for an adaptive user interface, we also report first feedback from developers regarding the desirability of such a user interface, which indicated that adaptations for development tools need to strike a careful balance between automation and user control. Nonetheless, the developers see the value in a gaze-based adaptive user interface and how it could improve software development tools going forward.

CCS CONCEPTS

• **Software and its engineering** → **Development frameworks and environments**; • **Human-centered computing** → *Empirical studies in HCI*; *Interaction design process and methods.*

KEYWORDS

human-computer interaction, adaptive UI, integrated development environment, software engineering, tools, eye tracking, gaze, automation

ACM Reference Format:
Thomas Weber, Rafael Vinicius Mourão Thiel, and Sven Mayer. 2023. Supporting Software Developers Through a Gaze-Based Adaptive IDE. In *Mensch und Computer 2023 (MuC '23), September 03–06, 2023, Rapperswil, Switzerland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3603555.3603571

MuC '23, September 03–06, 2023, Rapperswil, Switzerland

ACM ISBN 979-8-4007-0771-1/23/09…$15.00
https://doi.org/10.1145/3603555.3603571

1 INTRODUCTION

Developing software systems is a complex task. Developers not only have to maintain an overview of the large code base but also consider dependencies, documentation, errors during compilation- and run-time, and much more. Thankfully, tools to support developers with these tasks have come a long way, and modern Integrated Development Environments (IDEs) provide mechanisms to help with just about any aspect of the development process in a single application [49]. Compared to the alternative, which would be small dedicated tools for each step, this integration helps to eliminate human and technical errors when transitioning from one task to the next. However, the high degree of integration has led to IDEs becoming increasingly complex to being bloated with functionality [26]. Rarely anyone requires all the features, particularly for novices, as the sheer volume of panels, buttons, and parameters can be overwhelming [41].

When a user interface (UI) becomes highly complex, selecting the correct sub-system and working context for a task adds extra steps and mental load [26]. Here, adaptive UIs can support the user by

shifting some workload and responsibility to the system; therefore, the system must determine the user's intent in a given context [29]. Various cues allow allow this adaptation; researchers have already used some to create adaptive IDEs for software developers, e.g., typical workflows [43]. Based on this, they predict what a developer might want to do next and then support this step by easing access to it or highlighting necessary information. How developers want to approach their next challenge depends on their individual work practices and the specific task at hand. However, mental processes, users' intent, and areas of interest are typically inaccessible information to the system. Based on initial results by Zhang et al. [54], we argue that eye-tracking data can support adaptive UIs in determining the developers' intent.

In this work, we present a gaze-based adaptive IDE which supports developers using Visual Studio by adapting the layout of the IDE. Furthermore, we contribute a first assessment of the feasibility and perception by developers of such a system. In a data collection study (N=6), we first retrieve 180 minutes of interaction data (e.g., gaze data, IDE events) to understand non-adaptive code development with Visual Studio. Using our new data set, we train a neural network that predicts the most important layout window given the latest user interactions. With that, we developed a Visual Studio extension allowing real-time IDE layout adaptation. To assess the potential benefits of our Visual Studio extension, we conducted a first evaluation study (N=6) with software developers, which highlights that meaningful adaptations can be achieved even from a small data base. Moreover, we gain insights into the perceived benefits and drawbacks of layout adaptation and explore the desirability of adaptive UIs for software development.

Our results show that a human-in-the-loop gaze-based adaptation can support developers during coding. The key here is that the layout adaptation are not distracting for the developer, which can be achieved by infrequent and less abrupt adaptations. Our system, for example, adapted the IDE up to 2.3 times per minute, but developers reported that they perceived these changes only minimally. Additionally, developers are generally open to layout adaptations supporting them. These are promising first results as our gaze-based adaptation could support them while it did not distract them from their primary task. On the other hand, we also found that developers are skeptical when the adaptation would restrict functionality in a future version of this system. In summary, developers favor automatic adaptation as long as they have the opportunity to override it if needed; thus, they never restrict them in their development.

2 RELATED WORK

While adaptive UIs are an idea that has been around for a while, both, the application for software development tools and the use of gaze as an information source, has been explored only to a limited degree. The following section will outline some general background information and highlight some of the existing literature on adaptive UIs in this specific context.

2.1 Adaptive UI

The UI layer is a crucial component in software applications. Ultimately, it connects users to a software's functionality. So even a well-made software application might come short due to a suboptimal UI layer. Whereas some older user interface development techniques, such as *universal design* [30], *inclusive design* [25], and *design for all* [46], advocate a *one design fits all* approach, a user interface is dependent on its *context of use*, which can be "decomposed into three facets: the user, the computing platform [...] and the complete environment" [8]. Hence, it is hardly possible for one user interface to accommodate all the use cases in a given context of use, leading to a potentially diminished user experience [1] and less effective operation of the software application.

Building only a single, fixed UI forces the designer to predict all possible variants of usage – a task hardly possible. To mitigate this, one can instead design the UI to be *adaptive*. Such user interfaces with adaptive properties are occasionally referred to as multi-context or multi-target UIs [9], but this work will use the term *adaptive UI* for its endeavor to incorporate gaze data as a potential source of rationale into an adaptive user interface.

An adaptive user interface extends a regular user interface by adding adaptive elements and layouts [5], thus tailoring the presentation of functionality to a user's task at hand, personal preferences, or different contexts of use. This can potentially "reduce visual search time, cognitive load, and motor movement" [16]. In essence, an adaptive user interface attempts to predict what a user might want to know or do and assist them with it, so they have to spend less time and effort. This prediction can be defined by users in the form of rule-based systems [20, 27], fully automated using, e.g., Machine Learning [15, 28] or a hybrid human-in-the-loop system where user feedback steers the automated prediction [32, 43]. All these variants roughly follow the blueprint for a "dynamic [and] seamlessly personalized" [29] adaptive user interface as outlined by Liu et al. [29] where an adaptive UI executes the following functions: 1) Observe the interaction between the user and software application, 2) Identify distinguishable episodes, 3) Recognize user behavior patterns, 4) Help users according to recognized user plans, and 5) Build user profiles to enable personalized interactions.

To perform these tasks, an adaptive UI should "extract as much useful information [...] as possible" [29] and categorize that data into "episodes". Data to make this categorization can come from various sources, e.g., internal metrics from the software, user interaction with mouse and keyboard, or, as we further explore, physical properties of the user, like gaze. A sufficiently large and varied database allows the system to find "hidden patterns in the streams of events", which inform the adaptive functionality.

Whenever the system recognizes a pattern, thus being able to predict what the user is going execute, it can "offer assistance" adaptively. Lastly, the system can create user profiles to store relevant user-defined preferences and user-specific patterns to provide adaptions and personalization.

To further describe adaptive systems, Salehie et al. [42] presented a hierarchy of adaptability in general software systems. This hierarchy also applies to adaptive UIs (cf. Akiki et al. [1]). The hierarchy is as follows:

Context-awareness "indicates that a system is aware of its context, which is its operating environment" [42] Only if the UI is aware of its context, it is able to trigger adequate adaptations [1].

Self-configuring "is the capability of re-configuring automatically and dynamically in response to changes" [42] The context of use is not static; it is constantly evolving (for example, a user's computer skills improve). Therefore, the adaptation rules have to be kept up to date. This can be done with a mechanism that monitors changes in context; another alternative is the incorporation of user feedback.

Self-optimizing "is the capability of managing performance and resource allocation in order to satisfy the requirements of different users" [42] In the context of adaptive user interfaces, this can mean that a system can "self optimize by adapting some of its properties" [1].

Where an adaptive UI can be positioned within this hierarchy depends on several factors, e.g., the volume and variety of data. The more data a system has available, particularly real-time data, the more it can typically infer about the context and user and thus offer more complex adaptations. To this end, we explore gaze as an additional data source.

Eye tracking as a data source for adaptive UIs has been the topic of some research projects, e.g., [38] who utilized gaze data for an adaptive UI in an augmented reality (AR) setting to show or hide virtual information depending on whether the user looks at the real-world environment or the AR elements. Göbel et al. [21] also uses eye tracking to adapt a UI for displaying maps to hide irrelevant information and highlight relevant UI elements. An alternative to this is to rearrange UI elements to minimize load on the user, as done, for example, by Gebhardt et al. [19])

Zhang et al. [54] similarly used gaze as a means to infer users' intent for a text formatting task. Using gaze data alone, they were only able to achieve an accuracy of roughly 40%. While they did also use actions for intent classification to greater success, they did not yet combine multiple data sources and propose this as an interesting approach. Furthermore, their analysis, so far, focused on accuracy metrics, and did not assess adaptations based on intent prediction or the subjective perception of these.

As these examples show, using gaze data has proven an interesting and feasible source of data for adaptive UIs in select specific contexts and also to infer a user's mental state [11]. However, it appears gaze should be leveraged in combination with additional data sources [54]. In the following paper, we contribute to this by investigating an adaptive UIs using gaze and interaction data in the context of software development tools where existing tools offer a high degree of complexity, putting a lot of mental load onto the user [26, 41]. These UIs are often not very visual and rely heavily on the text, while many expert users often rely on shortcuts. A lot of the interaction thus happens during reading and thinking of the developer in front of the UI, which makes common data sources for adaptations like user input or prior-knowledge [48] less helpful on their own, thus making gaze an interesting additional data source.

2.2 UI Adaptations

The previous section described on which basis an adaptive user interface should make its adaptations. Now, how can the adaptive UI then translate those triggers into the visual layer and offer assistance? We follow Schmidmaier et al. [43] by distinguishing between different degrees of adaptation, ranging from adaptations of the visual presentation to adaptations of the underlying system functionality.

Changes to the presentation can start with very subtle adaptations. Providing contextual hints and tips is one fairly common example. Instead of showing additional information, an adaptive system can also do the opposite and hide information, which presents a slightly more serious intervention in the interaction. Ideally, this should only affect less relevant information and be easily reversible. Since this must not be a binary decision, the information is there or not, but can also mean that information is partially hidden or made less visible or intrusive, Schmidmaier et al. [43] refer to this as *Information Dimming*. The goal here is typically to "reduce cognitive overload and support task focus flow" [43].

Going beyond simple changes to the visual appearance, adaptations can also trigger or change system behavior. This is typically done in an effort to "dynamically enhance the user's workflow" [43], provide shortcuts, and increase efficiency. A system can determine what sequence of actions is typical for a user and then provide specific shortcuts that group these actions into a single command. Taking this one step further, when the system is able to predict what a user typically would do next, it could go ahead and execute those commands automatically (cf. Schmidmaier et al. [43]). To predict the next upcoming command, the system will require an abundance of contextual information, though, particularly in complex systems where adaptations make sense, the search space for potential commands tends to be quite large.

Such dramatic changes only make sense in a few circumstances and require the user to be very familiar with them so as not to be confusing, as they otherwise run the risk of having adverse effects on usability, which could negate the benefit of the adaptations.

2.3 User Acceptance

Adaptations, particularly the more aggressive interventions into the system's functionality like those mentioned above, can certainly reduce the time spent on tasks, but they can also increase the risk that it becomes unclear to the user why and when actions are executed and reduce the predictability of the system. "Disorientation and the feeling of losing control" [36] make it necessary that an adaptive UI offers additional mitigating support mechanisms to maintain a certain degree of understanding and control like post-hoc explanations, confirmation requests, "feedback and undo functionality" [43]. Whenever adaptations occur, the user has to be able to understand the automatic changes to the UI – it is important to support a "feeling of continuity between the UI before and after adaption" [14] and to show the user what caused the adaptation, leading to better predictability of the system behavior [24]. The implementation of a feedback loop can increase transparency and user acceptance. Furthermore, user feedback should not only be used for future adaptations but, if applicable, undo current adaptations as well to give back control to the user. Since incorrect adaptations can be considered an additional burden to the user [51], the benefits of correct adaptation have to outweigh the costs of those usability hiccups. Using user-centered methods during design and continuous user feedback should help reduce the average usability cost of an adaptive UI, making them more viable and beneficial.

2.4 Adaptive UIs in Software Development

Software devleopers, of course, are a different target group than general end users. Given their expertise with technology, they may be open to controlling an adaptive UI or may even expect this level of user control. There is, in fact, research into how software developers can facilitate and control UI adaptations, like the Adapt-UI system by Yigitbas et al. [52, 53] which uses the model-driven development to simplify the creation of runtime adaptations of UIs. To adapt the UI that the software developers themselves use, Schmidmaier et al. [43] propose an architecture, based on a combined approach where an offline trained model is refined using additional online learning. So far, most such systems that predict the developers behavior are limited in how the prediction is utilized [6, 13, 18, 34], though, typically in the form of action recommendations and not in automatic adaptations. These systems build on the interaction data, which is available directly from the development environment but do not yet draw on additional external sources of information about the developer, like gaze. In the context of software developers, eye-tracking has proven useful, though, yielding information about the behavior of software developers, e.g., code reading behavior [7, 37, 50], code comprehension [4, 40], or debugging [3, 23], so it is a promising additional source of information for UI adaptations. We will therefore explore how gaze data can contribute to an adaptive IDE.

3 ADAPTIVE IDE

While adaptive UIs can come in many forms, the goal is always to ease user access to functionality or information. In the context of an IDE, a UI that contains an overabundance of information and functions, we see two particular aspects of the UI that can be simplified by automatic adaptation: 1) timely access to relevant information and 2) choosing the correct functionality from the large range of IDE features. For this, we utilize the users' gaze and interaction data to adapt the UI's layout to make relevant information and features more visible. In detail, we developed a Visual Studio plugin, which adapts the layout of the IDE to support the developer during various tasks by changing the size of window elements.

3.1 System Design

We base the design of our prototype – as most adaptive systems – on two primary decisions: 1) what does change in the UI? and 2) what triggers these changes?

When deciding what **parts of the UI should be changed**, we considered that radical UI changes can also be distracting, particularly when UI elements change position [16, 33]. This is particularly unfortunate when the developer is in the middle of a lengthy task and is in a state of flow [35], where distractions may be worse than what is gained by the adaptations. Therefore, we decided to focus on information dimming and highlighting. Rather than radical changes in the UI, this means gradually removing less relevant or redundant information and making important information more salient.

Most IDEs continuously present much information unrelated to the user's current task and only becomes important in certain situations. Being ever present, the user may easily ignore or overlook this information when it suddenly becomes relevant. Thus, we focus on ensuring that this information is noticed adequately by changing the layout and dimensions of the IDE's panels to increase their visual presence. This way, information will grow gradually if the underlying system deems it increasingly relevant to a variable size, and no UI element vanishes or becomes inaccessible. By tuning our system, we also ensured that the main editor of the IDE, where mostly all user input and little output happens, remains sufficiently large, as we do not want to deflect from the IDE's main purpose, to write code. Layout changes are also fairly easy to implement, making it a good choice for early design explorations, and it is easy to determine whether an adaptation was actually performed.

We generally decided to limit the changes and only apply them in small increments to minimize the risk of the aforementioned unwanted effects. Other, more aggressive changes, like pop-ups, changes in color, or even animations to get the users' attention, are possible. Such more drastic changes are likely more subject to personal preferences, though, and would skew the perception of our users. When subdued changes already elicit positive feedback, then future studies can determine how much adaptation is acceptable. Likewise, combinations of different adaptations are a reasonable approach for real-world applications. However, this makes any systematic evaluation challenging, so we focused on layout changes for now.

Concerning **the trigger**, we wanted them to be triggered by the context of use and guided by gaze data. With contextual information, we had two options: a) manually define rules that map a given context to an adaptation, or b) automate this process and let the system infer the relationship using machine learning. Rule-based systems have the advantage that they are typically easier to comprehend and control. However, they require expertise and experience to define good rules, and it is unclear how exactly we can utilize gaze data for manual rules. Automation with a continuously learning system also offers added flexibility, where rules can be personalized on-the-fly and through continued usage. Thus, we chose the second choice, using Machine Learning for layout adaptation. Although, it remained a design consideration that adaptations should not occur randomly since this would likely distract the user and, thus, limit efficiency for their tasks. Instead, an optimal adaptation system triggers the adaptations, ideally so that the user can understand and expect the adaptations after a learning phase.

3.2 Data Collection Study

For our adaptive IDE, we focused on a time series of gaze and interaction data, which, given a high enough resolution, give a detailed context of the user's focus of attention and the usage of the IDE, respectively. Since no suitable data set to support such development was publicly available, we first collected a set of data enabling us to train a machine learning model for real-time layout adaptation.

Procedure. After welcoming the participating software developers and answering any open questions, we asked them to sign an informed-consent form and fill in a demographics form. Then we invited the software developers to perform a series of well-defined tasks using an instrumented IDE. The tasks engaged the users with

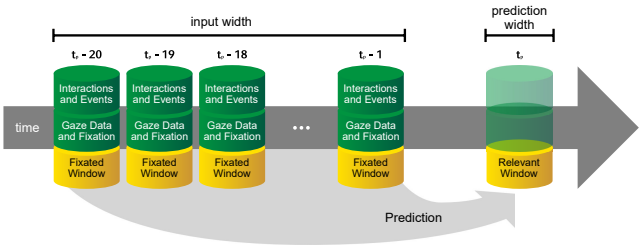

Figure 2: Our model uses the last twenty entries (input width) of interaction and gaze data, which are labeled with the fixated UI window, to predict a single step into the future (prediction width) which UI element will likely become relevant (also cf. [47]).

the IDE to ensure that our training data was meaningful and not just undirected, idle action. Therefore, we asked them to find and fix bugs in a set of seven sorting algorithms we provided, as this included both reading and writing code across multiple files. We did not give a number of errors in the code, but merely asked the participant to correct the code until it performs as specified, i.e., correctly sorting the input. The task description was always available on a secondary screen. Each participant had 30 minutes to do this. This time limit was chosen based on feedback from previous studies, in an effort to prevent exhaustion, which would negatively affect task performance, attention and thus potentially diminish the value of the gaze data.

Apparatus. Participants worked in a well-lit working space with a mouse, a keyboard, and a 27-inch screen with a 1920 × 1080 pixels resolution. We used a Tobii Eye Tracker 4C[1] to record the gaze. We calibrated the eye tracker for each participant. With the *iTrace* software [44], we could then match gaze to UI elements in the IDE window. Due to compatibility with the iTrace software, we exclusively worked with Microsoft Visual Studio 2017. To get a richer data set, we also utilized the Visual Studio API to extract the window's name, i.e., UI segment, on which iTrace detected the gaze. Beyond this, we used the Visual Studio API to capture the user's interactions with the IDE in the form of UI events.

Participants. Six participants contributed to the data set, yielding 180 minutes of gaze and interaction data with about one million data points. The participants were between 20 and 30 years old and had an academic computer science background and at least a bachelor's degree, with up to five years of professional software development experience. All of them completed the bug fixing task to completion in the given timeframe.

3.2.1 Recorded Data. We preprocessed the raw gaze data from our participants, mainly a standardization of the timestamps. Next, we extracted the gaze fixations; leaving all other gaze points in the data set could lead to information being hidden or highlighted, even though the triggering gaze was only a glance and not intentional attention. For this, we used the algorithm of the PyGaze [12] library to filter these gaze points and transform the raw gaze coordinates into fixation points. With continuous gaze turned into discrete time events, we could then pair each fixation with the corresponding UI events from the IDE. The combination of fixated UI event and actual interactions in the IDE should further reduce the likelihood of unexpected changes. The Visual Studio API constrained the kind of UI events we could collect. It provided us with four event types – Window Event, Command Event, Document Event, and Solution Event – each representing an interaction with the corresponding part of the IDE. Every event itself also holds additional properties and meta-information, for example, the triggering interaction and what element the target was. The Visual Studio API, iTrace [44], and FeedBaG++ [2] allowed us further to enrich each fixation with information from the UI elements. In total, each sample has 168 features values. So, we matched each fixation to one of the following elements:

- The **code editor window** that allows the user to write and change code inside an open file.
- The **output window** where a running program prints out logs and results.
- A window that allows the user to change **project-specific settings**.
- The **file explorer** from which a user can open or move files.
- A Visual Studio window that displays varying content depending on the IDE tools used by the user.

The combination of fixations from the gaze data, the meta-information for its corresponding UI element, and the UI events left us with a data set of about half a million sequential data.

3.3 Model

We trained a long short-term memory (LSTM) model with this data set to predict a suitable UI adaption using TensorFlow and Keras. We used a window size of 20 samples to predict the next important UI element, see Figure 2. While window size can be selected arbitrarily, we used a window size of 20, as it yielded good results and seemed to capture the current task. We trained the LSTM model to output a numerical importance score for each of the five possible panels.

After hyperparameter tuning, we found that 3 LSTM layers with 32, 64, and 32 neurons, followed by a dense layer as the output layer, performed the best. Moreover, we used a categorical cross-entropy loss function with a softmax activation function on the output layer. Any other parameters were left as their default value. As optimizer, we used the Adam optimizer with a batch size of 32, and we trained the model for 50 epochs. Finally, we used a time-based 70:30 train-validation split.

The final model achieved ~ 82% accuracy on the validation set. This model may leave room for improvement, but it does provide good results, given our validation set. The number of respective units and the amount of stacked LSTM layers may be altered in the future to optimize the model further.

3.4 Prototype

To incorporate this model into an IDE, we implemented a Visual Studio extension using the Visual Studio Extension API to access the IDE's internal event logging and the ability to change its behavior, in our case, the size and layout of the panels within the IDE window. A Python backend hooked up via a REST API is responsible for

[1] https://www.tobii.com

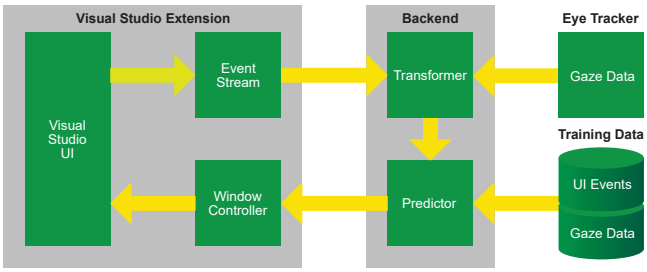

Figure 3: The prototype captures events inside the IDE and sends them to a backend via REST API. There the data is preprocessed, and a Machine Learning system predicts what UI elements are likely to be relevant. After sending this prediction back to the IDE, the IDE extensions adapt the UI accordingly, scaling the elements to match their predicted relevance.

incorporating the Machine Learning model, which decides when and how to change the UI. The expansion forwards any registered user input at the UI level to the backend. The backend uses 20 user inputs to generate a prediction and assess which UI element will be relevant next. The backend returns the prediction to the IDE, which adapts its user interface accordingly.

The whole architecture of our adaptive IDE Visual Studio Extension is illustrated in Figure 3. The Visual Studio Extension captures the *Event Stream* similarly to the FeedBaG++ tool [2]. The backend *Transformer* preprocesses the data for the Predictor. We pre-trained the *Predictor* with the *Training Data* from the data collection study. Whenever the *Window Controller* receives a prediction, it adapts the UI windows in size; in detail, it extends the default UI elements such that they can be stretched and shrunk at will (as showcased in Figure 1). Users can still manually change their size by drag-and-drop, and the *Window Controller* adapts their size relative to this manual preset. So, when the *Predictor* predicts one window of greater relevance, the *Window Controller* stretches this window and shrinks the other windows accordingly. If the prediction is ambiguous, the *Window Controller* promotes the code editor as a fallback solution. The changes made to the window sizes happen instantaneously without animation, so abrupt size changes are possible.

At this stage, even with the limited input data set, the prototype system can capture the user's context of use in real time and forward this information to a trained LSTM model that attempts to predict a window the user might consider relevant for the given context of use, and adapt the IDE's user interface to the predictions accordingly. The whole process takes no more than a few seconds and does not slow down the IDE's performance. While the technical feasibility is a good starting point, feedback from users is necessary to determine whether this technical solution also provides meaningful benefits in practice.

4 SYSTEM EVALUATION

We conducted a qualitative evaluation to assess whether our proof-of-concept already provides enough support that developers notice the adaptations and see their benefits. Since programming in an IDE is a highly creative activity where many ways can lead to a solution, comparability between software developers and programming tasks is often barely possible.

As personal experience with Visual Studio and its many features result in personal preferences, quantitative performance measures are hard to assess. Gajos et al. [17] highlighted this as they attempted to evaluate adaptive UIs quantitatively but found it challenging, while subjective and qualitative measures were more expressive. In contrast, qualitative feedback can still give a good impression of how people view the changes in our adaptive IDE generally, whether they see value in them and how the current prototype can be improved in the future into a production-ready tool. Therefore, we only collected qualitative feedback about the adaptations and their benefits. For this, we invited six new participants to use the adaptive IDE for a small set of tasks, after which we interviewed them about their experience. Additionally, we recorded the internal behavior of the adaptive component to be able to reproduce when and how often UI elements changed during the evaluation.

4.1 Procedure

After welcoming participants in person, we explained the purpose of the study and answered any open questions. Then we asked them to give written informed consent. Afterward, we asked them to sit down in front of a 27-inch monitor with a 1920 × 1080 pixel resolution, a keyboard, a mouse, and a Tobii Eye Tracker 4C (as in the data collection study). We presented the task as a Visual Studio project.

We asked participants to work on tasks in a controlled environment using the adaptive IDE with the eye tracker for 30 minutes. During the interaction, the adaptive IDE recorded and analyzed their behavior and adapted the UI accordingly. To avoid confusion, we informed the users of this behavior – after all, when someone would use an adaptive IDE in the wild, they too would know of this feature. The objective of the tasks was to keep the participants busy and interacting with the IDE, thus triggering a constant stream of events. In detail, we asked them to implement an algorithm for searching text and benchmark their code against a provided linear search algorithm. The success of the task itself was not the goal here. Instead, we designed the task to involve various interactions within the IDE, including writing and reading code, executing it, assessing the output, and making adjustments to ensure that different UI adaptations would occur.

After performing this task for 30 minutes, we asked the participants a series of questions in the form of a semi-structured interview. During the interview, participants had the opportunity to provide us with feedback and were encouraged to share their ideas with us for potential improvements.

4.2 Interview Guide

We wanted to know whether participants noticed any adaptations and how they perceived them. After getting a first impression, we moved on to more precise questions regarding the adaptations that occurred during the task, e.g., did they maintain their state for too long or too short? With a firm grasp of the capabilities of our prototype, we then asked to give specific feedback for our prototype, specifically what changes in the UI were helpful and which were detrimental, which further aspects of the UI they would like to see

adapted, and what their expectations are for long-term use. Moving from our prototype to adaptive UIs in general, we also discussed different types of adaptations and methods of personalization in IDEs and how the participants perceived them.

4.3 Participants

We invited six participants. Like the group for data collection, they all fell in the 20 to 30-year age group, and had at least a bachelor's degree in computer science and prior experience with professional software development. According to their feedback, all felt comfortable with the given code and had no problems with the task. All of them also had previously used an IDE, although not necessarily this version of Visual Studio, and were thus familiar with all the general UI elements which were present in our study.

4.4 Results

We used affinity diagramming [22] to label and cluster the statements of our participants into common themes and by their value judgment. For example, whether they perceived the IDE adaptation as positive or negative.

All participants noticed adaptations, although some only barely. Some appeared to remember some changes to the UI happening during their coding, and only in hindsight could they attribute those impressions to the adaptive IDE. However, even if they did not notice all of them, the extension logs revealed that each participant had between 5 and 70 adaptations in the 30 minutes. The participants were surprised when we confronted them with the absolute number of adaptations during the 30 minutes task, as they assumed a lower number. However, it is important to note that not all adaptions during the usage were extreme changes to the UI. Sometimes changes were subtle, short-lived, and incremental, which participants can easily miss due to change blindness [45].

The overall impression was fairly neutral when asked about their attitude toward the adaptations. The adaptations did not negatively impact the coding task or distract participants from their workflow. Moreover, participants agreed that adaptations were suitable for a given context. Finally, participants identified that the adaptations were suitable for the context in which they appeared. They could see the value of an adaptive, supporting UI regarding ease of use and efficiency.

Participants stated that to rate the adaptation positively, they needed to perceive the impact of the adaptation more clearly. Thus, the participants concluded that they observed too few adaptations and that a longer period of working with an adaptive IDE would be necessary to gauge their effect better. In contrast, we argue that perceivable adaptations are not desirable, as they most likely will distract from the development task itself.

The participants agreed that the worst-case scenario for adaptations in real-world use would be that they could hide relevant information, since the system erroneously considers other information to be more important. Fortunately, during our evaluation, this only occurred in a single instance. Here, the output window overlapped the code editor, hindering them from proceeding with coding. However, the participant fixed the issue quickly by adjusting the window layout. To account for such issues, another participant noted that a useful constraint could be to have the code visible at all times.

We also asked whether our participants would prefer smooth, animated transitions or more abrupt changes. Here, everyone except one participant agreed that they preferred the transition-less adaptations. They argued that animations could attract too much attention. Beyond our initial IDE adaptation, participants positively perceived UI adaptations but remained skeptical towards adaptations that influence the system's functionality. Many categorically stated that they did not want this adaptation in their IDEs but remained open to less intrusive adaptations like the layout adaptation. One participant was only willing to try adapting the functionality when extensive feedback and intervention mechanisms were in place to counteract the adaptations when necessary. The wish for more control and transparency was a common theme overall. One of the participants mentioned preferences with user-based priority weighting for supported windows. Several participants mentioned that predefined rules could also lead to better adaptations.

In summary, all participants showed interest in an adaptive IDE that has more to offer than what this proof of concept was capable of delivering. Everybody said they would try an adaptive IDE that highlights and hides information to see how it affects their work in the long run.

5 DISCUSSION

In this chapter, we discuss our findings with a focus on two key aspects: (1) our experience with eye tracking as a data basis for our implementation and the resulting learnings from implementing an adaptive IDE, and (2) the feedback from the users during the evaluation.

5.1 Gaze as Data-Basis for an Adaptive UI

We showcased the viability of using gaze data for an adaptive UI. Our prototype and the subsequent evaluation show that a functioning, adaptive UI is possible and even with the limited data set provides adequate adaptions. Our LSTM model could define and recognize usage patterns from the fixations. Our evaluation also showed that these patterns, e.g., highlighting of the output window, differed between the users, allowing for a personalized experience. The users, in turn, considered the adaptations suitable and unobtrusive for their context. Nevertheless, our implementation provided only a limited degree of adaptation, highlighting some panels and UI elements for users. However, this is a natural consequence of the relatively small data set we were able to use as input data. An industrial solution with wide adoption could utilize a more extensive initial set and incrementally increase its performance through online learning and real-world usage. On the one hand, it would allow training more complex models, yielding better accuracy. It would also need to support various use cases. In our data-collection and evaluation, we focused only on searching and sorting algorithms for which participants had to write or update code. While the adaptions we trained from one task and showed in another one were perceived as adequate, these task can already offer a wide variety. Of course, software development encompasses even more tasks and use cases and how people use their IDE can depend on

factors like the programming language or the phase in the development life cycle. Furthermore, the flexibility of IDEs allows for broad personalization beyond the fairly fixed state in our study. While most people will likely not completely alter the layout of their IDE for every task, it still highlights the challenge of tools as complex as IDEs and the broad range of starting conditions an adaptive UI will be able to cope with. So, for practical use and wider adoption, the underlying model needs to be expanded, which also requires additional data. The fact that even our small data set was able to generate meaningful adaptions, though, is promising, particularly for a personalized version of such an adaptive IDE, i.e., a use case where no large amounts of data will be available. Once a developer has configured their IDE, it would seem that training and personalization can happen quickly. particularly when using a combined online and offline learning approach [43]. The fact that a small data set was sufficient to show meaningful results is also relevant for future studies, where one might investigate other data sources. Not having to collect a large volume of data means that we can, in the future, quickly prototype variations of an adaptive IDE and evaluate them to get a better understanding which types of data are beneficial for timely and meaningful adaptations.

Window Size. As we learned during our work, individual gaze points or UI events can only be one part of such a data set, though, and are barely suitable in isolation. For example, when developers are searching for a solution or just thinking, their gaze may wander across the screen. Thus, interactions and gaze points may fluctuate over a short period. These fluctuations offer limited value compared to the larger context and longer periods of previous interactions. Thus, as was apparent during our work, it becomes a challenge to choose the right degree of granularity for both input data and adaptations to ensure that the changes in the UI are timely and context-sensitive but not overly sensitive and thus over-steering. In our prototype, we chose a window of 20 samples to aggregate a time series, which worked well for our use case. The exact effect of this aggregation window, i.e., whether larger or smaller frames would improve the adaptation quality, remains an open question, which warrants further investigation. The same goes for the variety of data, particularly how more data source like gaze will increase the quality or whether we will reach a point of diminishing returns.

Improving the Performance Further. Beyond this, our prototype also highlighted some issues with gaze as one such data source and the resulting adaptations. The most notable of these is a consequence of how we look at the world: while we may be cognitively focused on one thing, our eyes and gaze may wander, sometimes erratically. Furthermore, sudden jumps of the eyes are much more likely than for other input modalities, e.g., the cursor. Not only does this complicate predictions, but since patterns may be interrupted by completely irrelevant jumps, it also affects the resulting adaptations. If the input jumps frequently and these jumps affect the adaptations in the short term, the system may recommend barely noticeable adaptations because they are immediately overwritten and reset. In the worst case, this could lead to a jittery, flickering UI. Therefore, using a windowing approach is necessary. Moreover, our LSTM network architecture helps to reduce jittering.

Gaze and events are valuable sources on their own, and showcasing their feasibility for prediction is the core of this research. However, other data sources, such as the mouse, will be of value in improving the prediction, cf. [10, 31, 54]. On the other hand, expert users rely less on the mouse and instead use keyboard shortcuts. In addition, mouse movement tends to be very individual, c.f. [55]. Thus, in line with prior work Zhang, we argue that mouse movements will support the prediction, but a successful prediction will only be possible with gaze data. However, mouse data, in addition to gaze data, will further stabilize the prediction.

Comparison to Rule-Bases Adaptation. Rule-based adaptations could work around the challenges of fully autonomous adaptation. However, the granularity of gaze data makes it unfeasible to prescribe rules on the low-level data points manually. Automated classification methods and aggregation in conjunction with high-level rules that utilize the classified context could provide a hybrid solution. The human involvement then checks whether the aggregation works to determine the correct context. In such a scenario, users could then use this context to specify in which situations they want what changes and the system would primarily determine whether the conditions for an adaptation are met. We support such a hybrid approach, with feedback during our evaluation regarding rule-based, which was fairly positive. It would give additional control to the user, who can enable, disable, and change individual rules to suit their personal preferences. Increased user control could also facilitate a greater understanding of the adaptations, which prevents confusion due to sudden UI changes.

5.2 User Feedback

The feedback from our evaluation was primarily positive. Since all participants could perform the tasks we asked them to do, our prototype and its adaptions did not prove to be a hindrance. Our participants did not perceive the adaptive UI as distracting or interrupting.

Based on the feedback from the evaluation study, we conclude that implicit data sources like gaze can provide additional usage context and help with predictions. Nevertheless, their implicit nature makes them hard to control and understand for users. Therefore, it is essential to balance user control and automation. Thus, we argue that a hybrid system where automation determines the context and user feedback defines the behavior might foster enough trust that fully autonomous adaptation gains long-term adoption.

An objective performance metric certainly would be desirable, but this is challenging with a creative activity like programming and an artificial, short task in the lab. Long-term field observations may yield more insights into how adaptions in the IDE affect working behavior. For the subjective assessment of their performance, however, they did consider themselves to perform at least as good with the adaptions as without, so they proved not to be detrimental and potentially beneficial.

Longitudinal evaluations would offer further insights into which adaptions are most valuable. If adaptions have sufficiently impactful that they change the work practice of developers, we would also investigate how these behavioral changes interact with the adaptations, i.e., whether the initially trained adaptions remain adequate or whether the system needs to continuously improved, evolving adaptions and developer behavior alongside each other. Further, we chose only to investigate information dimming (adapting the

UI layout), but alternative adaptations are possible with our setup. Such studies inform the decision of which aspects of the adaptions can be fully automated and which ones users wish to control and manually personalize. Here, we see two options: a) the rule-based adaptations, where users explicitly define the system's behavior in a given context, and b) continuous human-in-the-loop feedback, where users provide immediate feedback when an adaption occurs.

However, we must keep in mind that the target group for an adaptive IDE, i.e., software developers, also does not represent the general public. It is very plausible to assume that those that develop software have a certain inclination to control it. They also have a higher degree of expertise in defining software behavior, so writing rules that map from usage context to system changes may be straightforward for them but not the average user. So, how these findings generalize beyond the context of our work will need to be tested, e.g. in similarly complex software where eye tracking already has shown potential like multimedia editing tools [39].

6 CONCLUSION

With this paper, we showed the viability of gaze data to turn an IDE into an adaptive UI. Using a relatively limited data set of gaze data, we could train and integrate a Machine Learning model, which changes the layout of the different IDE panels to match different contexts during software development. A first qualitative evaluation with developers showed that meaningful adaptations are possible and that the concept is overall well received, fostering an inherent value. At the same time, our participants wanted to retain a certain degree of control, so full automation based on gaze and interactions does not seem to be the final solution. Instead, hybrid approaches that utilize automation for context detection coupled with rule-based or human-in-the-loop feedback are our target group's preferred solution moving forward.

Our prototype acts as a starting point for future iterations and variations, as so far, we have only changed the layout of the IDE. More aggressive changes are possible, e.g., more direct highlighting of viable next steps, missed important information, active intervention, and automatically performing tasks that the developer typically does. Given the feedback we have gathered so far, such adaptive UIs will need to be evaluated in long-term, real-world studies to see how the adaptions and their usage evolve and whether the adaptations remain beneficial. These future investigations will yield exciting insights into how developers react to changes in their work.

REFERENCES

[1] Pierre A. Akiki, Arosha K. Bandara, and Yijun Yu. 2014. Adaptive Model-Driven User Interface Development Systems. *ACM Comput. Surv.* 47, 1 (2014), 9:1–9:33. https://doi.org/10.1145/2597999

[2] Sven Amann, Sebastian Proksch, and Sarah Nadi. 2016. FeedBaG: An interaction tracker for Visual Studio. In *24th IEEE International Conference on Program Comprehension, ICPC 2016, Austin, TX, USA, May 16-17, 2016*. IEEE Computer Society, New York, NY, USA, 1–3. https://doi.org/10.1109/ICPC.2016.7503741

[3] Roman Bednarik. 2012. Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations. *Int. J. Hum. Comput. Stud.* 70, 2 (2012), 143–155. https://doi.org/10.1016/j.ijhcs.2011.09.003

[4] Roman Bednarik and Markku Tukiainen. 2006. An eye-tracking methodology for characterizing program comprehension processes. In *Proceedings of the Eye Tracking Research & Application Symposium, ETRA 2006, San Diego, California, USA, March 27-29, 2006*, Kari-Jouko Räihä and Andrew T. Duchowski (Eds.). ACM, 125–132. https://doi.org/10.1145/1117309.1117356

[5] David Benyon. 1993. Accommodating Individual Differences through an Adaptive User Interface. *Human Factors in Information Technology* 10 (1993), 149–149.

[6] Tyson Bulmer, Lloyd Montgomery, and Daniela Damian. 2018. Predicting Developers' IDE Commands with Machine Learning. In *Proceedings of the 15th International Conference on Mining Software Repositories* (Gothenburg, Sweden) *(MSR '18)*. ACM, New York, NY, USA, 82–85. https://doi.org/10.1145/3196398.3196459

[7] Teresa Busjahn, Roman Bednarik, Andrew Begel, Martha Crosby, James H. Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm. 2015. Eye Movements in Code Reading: Relaxing the Linear Order. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension (ICPC '15)*. IEEE Computer Society, USA, 255–265. https://doi.org/10.1109/ICPC.2015.36

[8] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. 2003. A Unifying Reference Framework for multi-target user interfaces. *Interact. Comput.* 15, 3 (2003), 289–308. https://doi.org/10.1016/S0953-5438(03)00010-9

[9] Jose Manuel Cantera Fonseca. 2010. Model-Based UI XG Final Report.

[10] Mon Chu Chen, John R. Anderson, and Myeong Ho Sohn. 2001. What Can a Mouse Cursor Tell Us More? Correlation of Eye/Mouse Movements on Web Browsing. In *CHI '01 Extended Abstracts on Human Factors in Computing Systems* (Seattle, Washington) *(CHI EA '01)*. ACM, New York, NY, USA, 281–282. https://doi.org/10.1145/634067.634234

[11] Cristina Conati, Christina Merten, Saleema Amershi, and Kasia Muldner. 2007. Using Eye-Tracking Data for High-Level User Modeling in Adaptive Interfaces. In *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2* (Vancouver, British Columbia, Canada) *(AAAI'07)*. AAAI Press, California, USA, 1614–1617.

[12] Edwin S. Dalmaijer, Sebastiaan Mathôt, and Stefan Van der Stigchel. 2014. PyGaze: An open-source, cross-platform toolbox for minimal-effort programming of eye-tracking experiments. *Behavior Research Methods* 46, 4 (01 Dec 2014), 913–921. https://doi.org/10.3758/s13428-013-0422-2

[13] Kostadin Damevski, Hui Chen, David C. Shepherd, Nicholas A. Kraft, and Lori Pollock. 2018. Predicting Future Developer Behavior in the IDE Using Topic Models. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. ACM, New York, NY, USA, 932. https://doi.org/10.1145/3180155.3182541

[14] Charles-Eric Dessart, Vivian Genaro Motti, and Jean Vanderdonckt. 2011. Showing User Interface Adaptivity by Animated Transitions. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Pisa, Italy) *(EICS '11)*. ACM, New York, NY, USA, 95–104. https://doi.org/10.1145/1996461.1996501

[15] Steven M. Drucker, Danyel Fisher, and Sumit Basu. 2011. Helping Users Sort Faster with Adaptive Machine Learning Recommendations. In *Human-Computer Interaction – INTERACT 2011*. Springer Berlin Heidelberg, Berlin, Heidelberg, 187–203. https://doi.org/10.1007/978-3-642-23765-2_13

[16] Leah Findlater and Krzysztof Z. Gajos. 2009. Design Space and Evaluation Challenges of Adaptive Graphical User Interfaces. *AI Mag.* 30, 4 (2009), 68–73. https://doi.org/10.1609/aimag.v30i4.2268

[17] Krzysztof Z. Gajos, Mary Czerwinski, Desney S. Tan, and Daniel S. Weld. 2006. Exploring the Design Space for Adaptive Graphical User Interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (Venezia, Italy) *(AVI '06)*. ACM, New York, NY, USA, 201–208. https://doi.org/10.1145/1133265.1133306

[18] Marko Gasparic, Tural Gurbanov, and Francesco Ricci. 2017. Context-aware integrated development environment command recommender systems. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 688–693. https://doi.org/10.1109/ASE.2017.8115679

[19] Christoph Gebhardt, Brian Hecox, Bas van Opheusden, Daniel Wigdor, James Hillis, Otmar Hilliges, and Hrvoje Benko. 2019. Learning Cooperative Personalized Policies from Gaze Data. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. ACM, New York, NY, USA, 197–208. https://doi.org/10.1145/3332165.3347933

[20] Panagiotis Germanakos, Marios Belk, Argyris Constantinides, and George Samaras. 2015. The PersonaWeb System: Personalizing E-Commerce Environments based on Human Factors. In *Posters, Demos, Late-breaking Results and Workshop Proceedings of the 23rd Conference on User Modeling, Adaptation, and Personalization (UMAP'15 Workshop Proceedings, Vol. 1388)*, Alexandra I. Cristea, Judith Masthoff, Alan Said, and Nava Tintarev (Eds.). CEUR-WS.org, Online, 4.

[21] Fabian Göbel, Peter Kiefer, Ioannis Giannopoulos, Andrew T. Duchowski, and Martin Raubal. 2018. Improving Map Reading with Gaze-Adaptive Legends. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications* (Warsaw, Poland) *(ETRA '18)*. ACM, New York, NY, USA, Article 29, 9 pages. https://doi.org/10.1145/3204493.3204544

[22] Gunnar Harboe and Elaine M. Huang. 2015. Real-World Affinity Diagramming Practices: Bridging the Paper-Digital Gap. In *Proc. 33rd Annual ACM Conf. Human Factors in Computing Systems*. ACM, New York, NY, USA, 95–104. https://doi.org/10.1145/2702123.2702561

[23] Prateek Hejmady and N. Hari Narayanan. 2012. Visual attention patterns during program debugging with an IDE. In *Proceedings of the 2012 Symposium on Eye-Tracking Research and Applications, ETRA 2012, Santa Barbara, CA, USA, March*

28-30, 2012, Carlos Hitoshi Morimoto, Howell O. Istance, Stephen N. Spencer, Jeffrey B. Mulligan, and Pernilla Qvarfordt (Eds.). ACM, 197–200. https://doi.org/10.1145/2168556.2168592

[24] Anthony David Jameson. 2009. Understanding and Dealing With Usability Side Effects of Intelligent Processing. *AI Mag.* 30, 4 (2009), 23–40. https://doi.org/10.1609/aimag.v30i4.2274

[25] Simeon Keates, P. John Clarkson, Lee-Anne Harrison, and Peter Robinson. 2000. Towards a Practical Inclusive Design Approach. In *Proceedings on the 2000 Conference on Universal Usability* (Arlington, Virginia, USA) *(CUU '00)*. ACM, New York, NY, USA, 45–52. https://doi.org/10.1145/355460.355471

[26] Rex Bryan Kline and Ahmed Seffah. 2005. Evaluation of integrated software development environments: Challenges and results from three empirical studies. *Int. J. Hum. Comput. Stud.* 63, 6 (2005), 607–627. https://doi.org/10.1016/j.ijhcs.2005.05.002

[27] Sucheta V. Kolekar, Radhika M. Pai, and M. M. Manohara Pai. 2019. Rule based adaptive user interface for adaptive E-learning system. *Educ. Inf. Technol.* 24, 1 (2019), 613–641. https://doi.org/10.1007/s10639-018-9788-1

[28] Pat Langley. 1997. Machine Learning for Adaptive User Interfaces. In *KI-97: Advances in Artificial Intelligence, 21st Annual German Conference on Artificial Intelligence, Freiburg, Germany, September 9-12, 1997, Proceedings (Lecture Notes in Computer Science, Vol. 1303)*. Springer, Cham, Switzerland, 53–62. https://doi.org/10.1007/3540634932_3

[29] Jiming Liu, Kelvin Chi Kuen Wong, and Ka Keung Hui. 2003. An Adaptive User Interface Based On Personalized Learning. *IEEE Intell. Syst.* 18, 2 (2003), 52–57. https://doi.org/10.1109/MIS.2003.1193657

[30] Ronald Mace. 1991. Accessible Environments : Toward Universal Design. *Design Interventions : Toward A More Humane Architecture* 156 (1991), 2.

[31] James S. Magnuson. 2005. Moving hand reveals dynamics of thought. *Proceedings of the National Academy of Sciences* 102, 29 (2005), 9995–9996. https://doi.org/10.1073/pnas.0504413102 arXiv:https://www.pnas.org/content/102/29/9995.full.pdf

[32] Nesrine Mezhoudi. 2013. User Interface Adaptation Based on User Feedback and Machine Learning. In *Proceedings of the Companion Publication of the 2013 International Conference on Intelligent User Interfaces Companion* (Santa Monica, California, USA) *(IUI '13 Companion)*. ACM, New York, NY, USA, 25–28. https://doi.org/10.1145/2451176.2451184

[33] J. Mitchell and B. Shneiderman. 1989. Dynamic versus Static Menus: An Exploratory Comparison. *SIGCHI Bull.* 20, 4 (apr 1989), 33–37. https://doi.org/10.1145/67243.67247

[34] Emerson Murphy-Hill, Rahul Jiresal, and Gail C. Murphy. 2012. Improving Software Developers' Fluency by Recommending Development Environment Commands. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (Cary, North Carolina) *(FSE '12)*. ACM, New York, NY, USA, Article 42, 11 pages. https://doi.org/10.1145/2393596.2393645

[35] Jeanne Nakamura and Mihaly Csikszentmihalyi. 2014. The Concept of Flow. In *Flow and the Foundations of Positive Psychology: The Collected Works of Mihaly Csikszentmihalyi*. Springer Netherlands, Dordrecht, 239–263. https://doi.org/10.1007/978-94-017-9088-8_16

[36] Matthias Peissner and Rob Edlin-White. 2013. User Control in Adaptive User Interfaces for Accessibility. In *Human-Computer Interaction – INTERACT 2013*. Springer Berlin Heidelberg, Berlin, Heidelberg, 623–640. https://doi.org/10.1007/978-3-642-40483-2_44

[37] Norman Peitek, Janet Siegmund, and Sven Apel. 2020. What Drives the Reading Order of Programmers? An Eye Tracking Study. In *Proceedings of the 28th International Conference on Program Comprehension* (Seoul, Republic of Korea) *(ICPC '20)*. ACM, New York, NY, USA, 342–353. https://doi.org/10.1145/3387904.3389279

[38] Ken Pfeuffer, Yasmeen Abdrabou, Augusto Esteves, Radiah Rivu, Yomna Abdelrahman, Stefanie Meitner, Amr Saadi, and Florian Alt. 2021. ARtention: A design space for gaze-adaptive user interfaces in augmented reality. *Comput. Graph.* 95 (2021), 1–12. https://doi.org/10.1016/j.cag.2021.01.001

[39] Ken Pfeuffer, Jason Alexander, Ming Ki Chong, Yanxia Zhang, and Hans Gellersen. 2015. Gaze-Shifting: Direct-Indirect Input with Pen and Touch Modulated by Gaze. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 373–383. https://doi.org/10.1145/2807442.2807460

[40] Paige Rodeghero, Collin McMillan, Paul W. McBurney, Nigel Bosch, and Sidney K. D'Mello. 2014. Improving automated source code summarization via an eye-tracking study of programmers. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, Pankaj Jalote, Lionel C. Briand, and André van der Hoek (Eds.). ACM, 390–401. https://doi.org/10.1145/2568225.2568247

[41] Jean Michel Rouly, Jonathan D. Orbeck, and Eugene Syriani. 2014. Usability and Suitability Survey of Features in Visual Ides for Non-Programmers. In *Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools* (Portland, Oregon, USA) *(PLATEAU '14)*. ACM, New York, NY, USA, 31–42. https://doi.org/10.1145/2688204.2688207

[42] Mazeiar Salehie and Ladan Tahvildari. 2009. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.* 4, 2 (2009), 14:1–14:42. https://doi.org/10.1145/1516533.1516538

[43] Matthias Schmidmaier, Zhiwei Han, Thomas Weber, Yuanting Liu, and Heinrich Hußmann. 2019. Real-Time Personalization in Adaptive IDEs. In *Adjunct Publication of the 27th Conference on User Modeling, Adaptation and Personalization* (Larnaca, Cyprus) *(UMAP'19 Adjunct)*. ACM, New York, NY, USA, 81–86. https://doi.org/10.1145/3314183.3324975

[44] Timothy R. Shaffer, Jenna L. Wise, Braden M. Walters, Sebastian C. Müller, Michael Falcone, and Bonita Sharif. 2015. iTrace: Enabling Eye Tracking on Software Artifacts within the IDE to Support Software Engineering Tasks. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. ACM, New York, NY, USA, 954–957. https://doi.org/10.1145/2786805.2803188

[45] Daniel J. Simons and Daniel T. Levin. 1997. Change blindness. *Trends in Cognitive Sciences* 1, 7 (01 Oct 1997), 261–267. https://doi.org/10.1016/S1364-6613(97)01080-2

[46] Constantine Stephanidis. 1997. Towards the Next Generation of UIST: Developing for All Users. In *Proceedings of the Seventh International Conference on Human-Computer Interaction-Volume 1 - Volume I (HCI International '97)*. Elsevier Science Inc., USA, 473–476.

[47] Tensorflow. 2022. Time Series Forecasting.

[48] Kashyap Todi, Gilles Bailly, Luis Leiva, and Antti Oulasvirta. 2021. Adapting User Interfaces with Model-Based Reinforcement Learning. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. ACM, New York, NY, USA, Article 573, 13 pages. https://doi.org/10.1145/3411764.3445497

[49] Thomas Weber and Heinrich Hußmann. 2022. Tooling for Developing Data-Driven Applications: Overview and Outlook. In *Proceedings of Mensch Und Computer 2022* (Darmstadt, Germany) *(MuC '22)*. Association for Computing Machinery, New York, NY, USA, 66–77. https://doi.org/10.1145/3543758.3543779

[50] Thomas Weber, Christina Winiker, and Heinrich Hussmann. 2021. A Closer Look at Machine Learning Code. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI EA '21)*. ACM, New York, NY, USA, Article 338, 6 pages. https://doi.org/10.1145/3411763.3451679

[51] Daniel S. Weld, Corin Anderson, Pedro Domingos, Oren Etzioni, Krzysztof Gajos, Tessa Lau, and Steve Wolfman. 2003. Automatically Personalizing User Interfaces. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence* (Acapulco, Mexico) *(IJCAI'03)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1613–1619.

[52] Enes Yigitbas, Stefan Sauer, and Gregor Engels. 2017. Adapt-UI: An IDE Supporting Model-Driven Development of Self-Adaptive UIs. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Lisbon, Portugal) *(EICS '17)*. ACM, New York, NY, USA, 99–104. https://doi.org/10.1145/3102113.3102144

[53] Enes Yigitbas, Hagen Stahl, Stefan Sauer, and Gregor Engels. 2017. Self-adaptive UIs: Integrated Model-Driven Development of UIs and Their Adaptations. In *Modelling Foundations and Applications*, Anthony Anjorin and Huáscar Espinoza (Eds.). Springer International Publishing, Cham, 126–141.

[54] Guanhua Zhang, Susanne Hindennach, Jan Leusmann, Felix Bühler, Benedict Steuerlein, Sven Mayer, Mihai Bâce, and Andreas Bulling. 2022. Predicting Next Actions and Latent Intents during Text Formatting. In *Proceedings of the CHI Workshop Computational Approaches for Understanding, Generating, and Adapting User Interfaces*. University of Stuttgart, Stuttgart, Germany, 1–6.

[55] Nan Zheng, Aaron Paloski, and Haining Wang. 2016. An Efficient User Verification System Using Angle-Based Mouse Movement Biometrics. *ACM Trans. Inf. Syst. Secur.* 18, 3 (2016), 11:1–11:27. https://doi.org/10.1145/2893185