Using Consumer Electronics to Enhance the Experience of Developing Software With Additional Input Modalities

How can we use a combination of gaze- and speech interaction to aid software developers?

Christoffer Hauthorn
201390049
201390049@post.au.dk

Jeppe Reinhold
201405981
jeppereinhold@gmail.com

Gustav Wengel
201407206
gustavwengel@gmail.com

Silas Roswall
201408919
201408919@post.au.dk

ABSTRACT

This paper demonstrates how consumer grade eye gaze tracking hardware and speech recognition software can enhance the software development experience for able-bodied software developers. Instead of replacing existing modalities, this paper explores the qualities of adding modern gaze tracking and voice recognition to the existing input modalities of the desktop. Through evaluation with computer science students, this paper shows that adding these input modalities compliments existing modalities, and enhances the user satisfaction when developing software.

Keywords

Speech Input; Eye Gaze Tracking; Multimodality; Software Development; User Satisfaction

1. INTRODUCTION

Different modalities have different strengths. When the mouse was introduced it helped increase the bandwidth in human computer interaction, and allowed for precision pointing. Both voice and gaze input have been researched before, but have mostly used expensive research grade electronics or concluded that the technology was not ready yet. The technology has evolved rapidly in the last few years, thus it makes sense to explore if the input bandwidth can be increased with consumer grade electronics.

This study focuses on enhancing the experience for able-bodied software developers using an Integrated Development Environment (IDE) to complete their work. We consider eye gaze tracking (EGT) hardware and voice recognition software to be at a price point that makes it relevant to professionals. This study therefore focus on user satisfaction when using the system in a desktop setting that resembles a regular (office) workstation.

Different solutions were discussed and ideated upon based on the related work. After reaching a shared vision for what was desirable and feasible, quick prototypes were created to evaluate the different modalities and their strengths on their own. Based on the findings from the initial evaluation, a final prototype integrating both modalities was designed, implemented and evaluated.

Students from Aarhus University were recruited to participate in the evaluations. The students were asked to complete computer tasks based on exam sets from the Introduction to Programming course[1] to ensure the assignments would be of similar complexity and difficulty.

The goal was to explore how to introduce the gaze and speech interaction, without interfering with or replacing already established modalities. We chose this approach to examine if it was possible to create a richer interaction with a greater number of modalities while maintaining or improving user satisfaction, as this is largely unexplored in research.

A video-demonstration of the final prototype can be seen at the following url: https://youtu.be/8wgXM70eQag

2. RELATED WORK

2.1 Eye Gaze Tracking

EGT has been studied for many decades in an attempt to further enrich Human-Computer Interaction for able-bodied people, and to enable disabled people in interaction with computers. Much of the focus have been on performance factors such as effectiveness and error rates. Studies have claimed that EGT is less accurate than a mouse as a selection device, requiring solutions such as dynamic zooming or greater target size and spacing to compensate for the lack of accuracy. Ware and Mikaelian [18] evaluates EGT from a performance perspective using 3 different selection techniques: dwell time, button press and an on-screen button. They conclude that EGT can be used as a fast selection device, if the targets are not too small. Dario et al. [13] uses a probabilistic model to more accurately calculate the

[1]At the Department of Computer Science at Aarhus University - http://kursuskatalog.au.dk/en/course/64210

Aarhus Universitet Bachelor Project 2016

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

users gaze position, along with an extended WIMP interface they call Intelligent Gaze-added Operation system. The main point of the IGO is to make targets bigger and with more spacing, to account for the lower accuracy of the gaze-tracking technology. Bates et al. [1] explores using gaze-interaction in a standard WIMP interface, through real-world computer tasks like browsing and word processing. They conclude that the system needs to support dynamic zooming because of the inaccurate gaze-tracking. They argue however, that the inaccuracy not only lies in the tracking technology, but also in the fact that the human eye are not always looking directly at the point they are reading, because the eyes can see objects that are close to the focus, instead of directly at the focus. Kumar et al. [7] also presents a system combining keyboard and gaze tracking for accurate on-screen pointing. They too utilise a zooming mechanism triggered by a button press to accurately select targets, and conclude that EGT in combination with other modalities can devise an intelligent interaction model, that utilizes EGT to infer user intention and attention. Within the last 2-3 years eye-tracking technology have seen improvements that allow for both cheaper, more reliable and more precise technology. Tobii has recently entered the consumer market for EGT market with their EyeX gaze-tracking bar[2].

2.2 Multiple Modalitites

Studies in multimodal interaction isn't a new phenomena either. Speech and gestures in particular have been explored extensively. The question whether users can or will use the extra modalities have been examined. Studies points towards a willingness to adopt multimodal input, and furthermore that multimodal input can reduce errors and execution time. Despite all the research and technological advancements, many researchers still find themselves restricted by the technology available. Oviatt et al. [12] explore how to introduce multimodality systems. They found that users often mixed multimodal and unimodal commands. Furthermore as mentioned elsewhere [11], users also have a tendency to prefer multimodality when faced with spatial depending interaction eg. maps. Oviatt et al. [12] furthermore claims that introducing multiple modalities to systems will result in complementary functionalities rather than introducing redundancy. Cohen et al. [3] explore how multimodal interaction can benefit a military scenario planning system, by integrating pen and speech input. They conclude that the multimodal interface allowed the use of more natural language than unimodal, and furthermore allowed for executing more complex tasks using simpler commands, which could be remembered by semantics rather than icons or menus. The error rate also dropped with up to 50% compared with unimodal systems. Richard A. Bolt [2] also explore how multimodality allows for using eg. pronouns to execute commands rather than relying on remembering every object on the computer. He eventually became restricted by the technology at hand, resulting in workarounds for using speech such as having the participants stop in the middle of a sentence to allow the system to catch up and correctly translate the inputs.

2.3 Speech Interaction

The research combining speech input with traditional mouse-keyboard input is however much more limited, and authors like Kristensson [6] primarily focus on naturally switching between the two modalities for the same input, as to improve learnability by being able to use the modality that is most comfortable for any given task.

[2]Tobii Eye X: http://tobiigaming.com/product/tobii-eyex/

2.4 Speech and EGT Combined

There's an extensive body of research combining speech and gaze tracking, but the research is mainly for disabled users, and seek to provide full GUI automation. Some research suggests using speech and EGT to infer context and tasks, while still using traditional mouse and keyboard to perform more complex operations. Mazur et al. [4] has recently proposed a solution for disabled users, which combines gaze- and speech-interaction using consumer-grade electronics. They conclude that more quantitative evaluation needs to be done to fully know the effect of the proposition. Van der Kamp and Sundstedt [16] examines the possibility of using EGT and speech interaction in a paint-application for hands-free drawing. Their findings are, that even though the combination offered less control than traditional input devices, the participants reported that it was more enjoyable. Zhang et al. [19] Combines EGT and speech to correct errors that occurs when these two modalities are used separately. The results indicates that combining EGT and speech technologies yields better results in term of errors when compared to using either of them individually. Research combining gaze and speech with keyboard input is very limited, but there are some examples, most notably Wang [17], combining several modalities including speech, gaze and keyboard into one system, to navigate in 3-Dimensional space. Wang's work shows that there are multiple interesting interaction techniques at play, when combining EGT, speech and traditional mouse-keyboard interaction. The user can choose the modality that suits his needs in a particular task and context. The system can also infer context and task based on all the modalities, allowing the user to complete complicated tasks without specifying the context or operation mode.

To sum up, much of the related work in the field shows that combining speech and EGT interaction can provide greater user experiences by enabling a richer interaction. Furthermore, a bigger portion of the research has gone into examining quantitative performance factors rather than qualitative user experiences.

3. ANALYSIS

Findings from some of the research described in the previous section indicates that speech and EGT among other things can be used to infer the user's intention and context, and we tried to evolve this idea further. As a part of this, we've attempted to uncover the strengths of the different modalities. We've examined whether consumer grade EGT has been developed enough to use for precise pointing, and explored different ways to use speech in combination with gaze to offer rich, contextual interaction.

During our initial research into the field we came to the realisation that there have been a greater focus on performance factors like accuracy and error rate using Fitt's Law [5], rather than on qualitative factors such as user satisfaction, engagement or use in relevant contexts. We therefore wanted to examine factors other than performance, and our main focus have been on how adding gaze and speech modalities affect user experience.

It is our belief that the technology surrounding speech recognition has seen greater adoption in the consumer market for the past few years in the form of products eg. Apple Siri[3], Amazon Alexa[4] and Google Assistant[5]. The trend is the same regarding EGT with consumer products eg. the Tobii EyeX. We want to explore if these consumer-grade technologies are ready to be used in professional environments or if they still have limitations that makes them unusable in the long run outside of the research labs.

As Bates and others argue, EGT hardware requires dynamic zooming or similar to accurately pick small targets in a gui. The context of the software developer therefore seems suitable to our chosen technology, because it doesn't rely extensively on accurate pointer movement. The content does require memorization of complex command and syntax, which [3] argues lends itself to multimodality using speech and gaze.

4. FIRST ITERATION

4.1 Design

After analysing related work, we explored how and if multimodal interaction would enrich users interaction with the computer. As written in the analysis, the use cases that was found most suited for each modalities was respectively context selection with gaze and complex command execution with speech.

4.1.1 The Design Process

The design process started out with a brainstorm to explore the design space surrounding the problem. Ideas like selecting menu items with the gaze-interface was thought up, but was later abandoned because of the relative inaccurate EGT, that wouldn't easily allow for selecting small objects. The ability to use speech-recognizition as direct speech-to-text (STT) typing was briefly discussed aswell. However the abnormal syntax and wording used in software development made us realise that this was a less obvious solution. Questions like *'How would the STT handle brackets, indentation and casing?'* arose, which was hard to come up with solutions for. After multiple brainstorms, the ideas was evaluated against the observations of current practices to find the solutions that best suited current workflows. As the focus of the study was also to evaluate current consumer technologies, these were also kept in mind when choosing the ideas that deserved further examination.

4.1.2 Envisioning

Wang [17] talks about how we can infer humans intention from where they are looking, if this combined with naturally spoken commands could result in a more natural, spoken interaction allowing the HCI field to go from inputting through arbitrary commands and tedious traversing through files and sub menus to simple thinking out loud what you want done and where. The final product would allow the software developers to look at an error message and say "google it", change focus between different windows across all monitors, switch between tabs in the browser or IDE just by looking at them and calling every possible command or menu selection in the IDE just by saying what he or she wants done. Basically completing complex and sometimes context-aware tasks just by thinking out loud while looking where the action is intended to take place.

4.1.3 The Outcome

The vision was then broken down into smaller steps, allowing for small iterative loops towards the end goal, since new knowledge gained along the way would most likely alter the first imaged vision.

Concerning context selection with gaze, a complete implementation would allow the user to select context across all programs and monitors. This full implementation wasn't made, given that the technology didn't allow for gaze recognition on several monitors. Since complex command execution was chosen for speech, speech to text interaction was left out of the implementation, leaving the focus on selecting complex commands. The structure of these are inspired by the ideas for contextual commands in the work by Bolt [2], though we decided to leave the "there" out of the commands for brevity and ease of use. Optimally all modalities would work together and across programs and monitors, but since this is a largely unexplored field for testing, a quick fail-fast approach was chosen, resulting in testing the extra modalities separately and only implementing them in use of an IDE. This decision made it possible to try out the technology fast and catch initial shortcomings with each modality that wasn't anticipated and might drastically alter the solution.

In the first iteration the gaze interface allowed the participants to switch between three editor panes[6] to test how context selection through gaze would work. The speech interface provided the user with six commands, which made it possible to see if the user would pick speech commands over keyboard shortcuts or mouse as selection. Both implementations allowed for a fast first iteration.

One of the main functions that was envisioned for the gaze modality, was to react on where the user was already looking, and to use that in context and thus eliminating the need for using different shortcuts to navigate to another view or depend only on mouse or keyboard to move the courser. Even though gaze interaction has been around for a while, it hasn't been widely adopted, furthermore using it as an extra modality is quite new and even less adopted. Therefore it needed to be evaluated if the user could adopt the extra modality and utilise it or if it would simply become a cognitive burden or completely ignored. Thus the first implementation allowed the user to switch between the different editor panes. This simple use case was designed to see if the user would welcome this new form of interaction with the computer.

The main function of the speech modality was to aid the user in executing different task that normally would require the use of different shortcuts, templates etc. Which would likely reduce the cognitive load of the user. Instead of remembering what you want to do and then remember the arbitrary key binding for it, the user would only need to remember the name like *'run tests'* instead of $shift{+}alt{+}f10$. Speech interfaces aren't new, people have used assistants

[3]http://www.apple.com/ios/siri/
[4]https://developer.amazon.com/alexa
[5]https://assistant.google.com/

[6]An editor pane, is a view displaying the selected text-file. More panes can be open at the same time, allowing the user to view and edit multiple text-files at the same time.

such as Siri, Google Assistent, Alexa etc. but as mentioned before, using it as an extra modality and not a substitute is not widely adopted. Thus again, we wanted to evaluate how much the user would use it in the context.

4.2 Evaluation

The first iteration of the product consists of two separate evaluations. One evaluating the proposed eye-tracking interaction, and one evaluating the speech interaction. The two evaluations were completely separate and carried out independent of each other, but with the same overall structure to follow. Both evaluations were performed with two different subjects (4 in total).

To compare the extra functionalities offered by our prototypes with the standard functionality in the IDE, the whole process described below was conducted one time with, and one time without the extra functionality. As recommended by [14] the order of evaluating with and without extra modalities was reversed for the other half of the subjects, to account for the order effect, also called counterbalancing in psychology [15]. This is often done to produce reliable data, and even though that wasn't the purpose of this evaluation, it gave insights into this type of evaluation that could be used for the final evaluation.

The evaluation was started with introducing the subject to the concept, the overall idea, and how the evaluation would proceed. One evaluator was assigned to taking notes, while the other carried out the conversation. This setup was used throughout the whole evaluation. Afterwards, the participant was introduced to the coding task. All participants were somewhat familiar with the task, as it was similar to an exam in a course they had all completed within the last 3 years. This meant that they were familiar with the overall structure of the task, but not the details. The task is to create simple relations between multiple classes in Java and is scheduled to take around 30 minutes.

Then, the participant was introduced to the prototype (either speech or gaze), how it worked, what it could and couldn't do. In case of the speech evaluation, the user was given a list of the commands he could issue, and this list was present with him throughout the evaluation. Then, the prototype was calibrated using the respective calibration tools. This contributed to giving the subject an understanding of the level of precision the prototype provided. As learnability [14] wasn't a focus in the evaluation, the user got 5-10 minutes to try the different prototype functionalities, with aid from the evaluators. Afterwards the coding task went underway. The screen was recorded during the task using screen capture software[7], for further analysis if needed. The setup is shown in figure 1.

The evaluation was rather informal, as the subject was allowed to ask the evaluator questions about the prototype, the task or other aspects of the evaluation. The purpose of the evaluation was not to evaluate on quantative parameters like effectiveness, but rather to understand how the user feels about the interaction, and explore the possibilities the prototype enables. Therefore, the trade-off between making non-identical, non-quantifiable evaluations and getting deeper insight into the participants thoughts was a trade-off that supported the purpose of the evaluation.

When the subject finished the task, he answered a questionnaire related to the task. The questionnaire was a collection of scoring questions, requiring the subjects to score different aspects of the prototype. The answers were recorded orally, in an attempt to encourage the subject to elaborate on their answers. Besides the questionnaire, an unstructured interview was carried out lasting around 10 minutes. The questionnaire along with the interview was done to gain insights about user satisfaction, which along with usability, was the main focus of the evaluations.

[7]https://www.movavi.com/screen-recorder/

4.2.1 Speech evaluation

For this evaluation, the functionality consisted of the user being able to say a predefined set of commands (see Section 4.1), and the system performing an action in response. The participants were introduced to the speech-interaction and it's functionality, and they were handed a list of all the possible commands they could issue, so they could remember them. The list consisted of the following commands:

1. Creating a constructor in a class
2. Printing something to the terminal, (System.out.println())
3. Creating a toString method
4. Creating a main method
5. Running the main method
6. Creating a new class

Although both participants expressed that they liked using the system with voice, there were still some trouble. The participants weren't expert users and thus didn't use many shortcuts during their normal work. Therefore the participants didn't know shortcuts for common methods like System.out.println() (Print string to console) or creating a main method. Therefore they valued e.g. the voice shortcut for printing greatly, even though it might be less efficient for more experienced users, than the shortcut.

The autogenerated code was found useful for avoiding mistakes. One of the participants forgot to include the mandatory 'args[]' in the main method, which meant it did not work. This is not a problem with autogenerated code.

Furthermore there were some perceived usability improvements. One participant knew the shortcut for running the main method existed, but did not remember it, because he disliked this particular shortcut, so he always defaulted to using the mouse. He preferred using speech instead of an arbitrary keyboard shortcut.

On the negative side, error handling was poor. One of the participants tried to create a constructor, but failed since the cursor was outside the class. Since the system works by emulating user shortcuts, there's no possibility to intelligently handle errors, and the system can behave unpredictably. It took researcher intervention for the participant to figure out what he was doing wrong.

The two main take aways from the initial speech test was the following: First, the user might know what to do, but don't know the exact way or shortcut to do it, thus making it easier for non expert users to just say what they want to do. Second, error handling is at the moment poor, and would have to be considered in a final solution.

Figure 1: Evaluation setup for speech interaction

4.2.2 *Gaze evaluation*

Prior to the gaze evaluation, some setup and limitations had to be done on the computer to make the prototype function properly. The IDE was split up into 3 editor panes, each containing one of the Java classes required to complete the task. This was a technical requirement for this early prototype, in order to be able to select the correct editor pane based on the participants gaze (figure 2). The participant could at any time press the keyboard shortcut Ctrl + F1 to change focus to the editor pane he was looking at. The prototype would calculate which pane was appropriate to change focus to, and issue keyboard shortcuts to IntelliJ IDEA to change focus. There was a slight delay of 250 milliseconds from pressing the keyboard shortcut before the focus was changed due to this implementation. The participants where limited in their customization of the editor panes, as the technical implementation relied on predefined values at which the 3 panes where located.

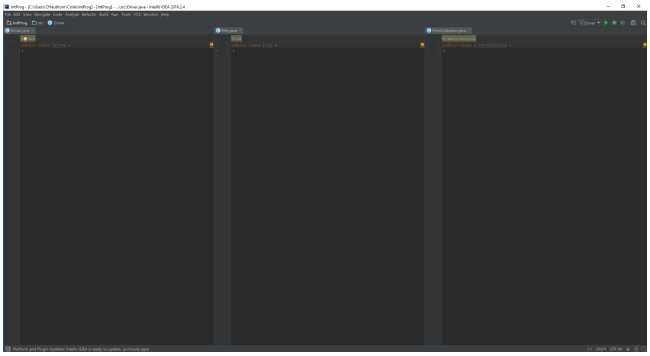

Figure 2: The initial gaze-evaluation setup with 3 editor panes

The Tobii EyeX[8] was mounted below the screen. Before the evaluation the participant was placed in front of the screen, and were introduced to the functionality of the tracker and the setup was calibrated for the particular participant.

Both participants reported some trouble adjusting to the keyboard used for the evaluation, as it was smaller than what both participants normally use. They also reported that the keyboard shortcut chosen to engage the pane switching was impractical and awkward to use, as it required stretching the hand and accurately press to buttons and the same time. Only one of the participants reported that they actually noticed the slight delay when pressing the shortcut, but that it didn't seem to affect his performance. When the second participant was asked to comment about this, he reported that he didn't notice the delay at all. Both participants reported that they enjoyed switching panes via gaze, and that this was preferred to using the mouse. They were used to pressing a keyboard shortcut multiple times to get to a specific editor pane, but the prototype allowed them to only press a shortcut once, to switch focus to any pane, regardless of position. During the post-task interview one participant said:

> This is at least as good as regular mouse and keyboard, and probably even better, although I don't know if there is a novelty effect that will wear off eventually.

At one point it was observed that a participant tried to write something where he was looking, and instantly realising that the cursor weren't where his eyes were looking. This could suggest that the system is quite appropriated by the user, even though it didn't even have this functionality the user tried to use.

Another observation was, that the participants spent a big amount of time navigation the code with the arrow keys, even over large distances. This inspired us to implement a 'Go' feature, that would instantly move the caret to the gaze-position at a press of a shortcut. The user would then have two different precision levels, one that switched focus to a given pane, and one that switched focus to a given point.

One of the participants suggested to implement the option to hold down Ctrl and then be able to press F1 multiple times in succession in order to quickly change pane again. He argued that some times he would mistakenly change to one pane and then realise that the next piece of code should go somewhere else.

The participants answered the questions from our questionnaire while the other participant was in the same room, which seemed to affect their results, as they argued for the same answers. It was therefore decided to limit the interaction between participants to a minimal in the final evaluation. Making the participants answer the same questionnaire two times (with and without gaze interaction) also created problems, as both participants gave the highest scores on almost all questions, making it hard to compare the two solutions with each other. Based on this finding, it was decided to combine the two questionnaires into one for the final evaluation, asking instead the participants to compare the two solutions rather than scoring them separately.

Overall, the feedback from the participants indicated that the gaze-interaction was promising, and there was merit in continuing work on it. The biggest problems that was observed doing the evaluation was mainly related to other factors than the gaze-interaction, such as a smaller keyboard, a wrongly chosen keyboard shortcut or the shortcut behaving a differently than expected. The evaluation led us to expand the feature set, change the evaluation setup and redo the post-task interview and questionnaire.

[8]http://tobiigaming.com/product/tobii-eyex/

5. SECOND ITERATION

5.1 Design

The goal of the second iteration was to get closer to the envisioned product, thus after testing each modality separately, the next step was to combine the two prototypes, to evaluate if this would actually make the process of software development more semantic and natural. The functionally in the second iteration with combined modalities is described in the following section.

5.1.1 Functionality

Jump to editor pane marks the pane the user is looking at, as the active pane in IntelliJ[9] IDE, allowing the user to continue where his caret was, last time he was in that pane. This can be activated by a keyboard shortcut. This functionality was kept since both participants in the first evaluation found it more useful than mouse or keyboard. This is combined with the voice commands.

Go to gaze moves the user's caret, to where he is currently looking. This can be activated by a keyboard shortcut, or a voice command "go". Thus building upon "jump to editor pane" allowing for further navigation without the need to remove the hand from the keyboard or tapping multiply times on the arrow keys. This also serves to test how well consumer-grade EGT performs with more precise pointing tasks. While it was not expected that the EGT would be precise enough to look at the exact area the user was looking at, the hope was that it would be precise enough to get close to the area the user was looking at, and allowing the final adjustments to be made via keyboard.

Speech commands is implemented in two different variations: *Regular* and *gaze-aware* commands. A regular command is a command that is executed with no regard for the location of the user's gaze. An example of a regular command could be "print" that simply tells the IDE to write *System.out.println()* where the caret is currently located. Gaze-aware commands are dependant on the current gaze-location and executed in the Java class that the user is looking at. An example of a gaze-aware command is *'new method'*. This command switches to the Java Class the user is looking at, and creates a new, empty method at the bottom. The user can then continue to edit the method with the keyboard. This is inspired from Wang's[17] work, where users intent can be inferred from where they a looking, eliminating the need to explicit say where the action is intended to take place. By combining a naturally spoken command with where the action is intended to take place, we hope to get closer to an interaction where the user just thinks out loud, and the machine seems to perceive what he wants to achieve.

Error handling was found to be problematic in the first iteration, since a participant accidentally placed the caret outside of the Java class and tried to create a constructor. By combining the two modalities and thus controlling how the user switches between panes, we aim to reduce this problem.

The functionality mentioned above was seen as the essential parts of the system that would allow us to examine the proposed research question, because code editing is the primary activity of software development. It is our hope that the combination of semantically meaningful commands and contextual data is one way to make the experience more satisfactory. Other functionality like debugging and file navigation using gaze and speech was also considered, but require precise pointing in the existing, and was therefore not designed and implemented.

5.2 Implementation

The final prototype consists of multiple independent pieces of software: The program using the Tobii EyeX framework to track the users gaze (called *Tobii EyeX WinForms Client* in 3), the python scripts to interface with Dragon NaturallySpeaking[10] (called *Dragon module* in 3), and of course IntelliJ IDEA, which is the program that receives mouse and keyboard input simulated from either the Tobii EyeX WinForms Client or from the Dragon module. See the overall architecture in figure 3.

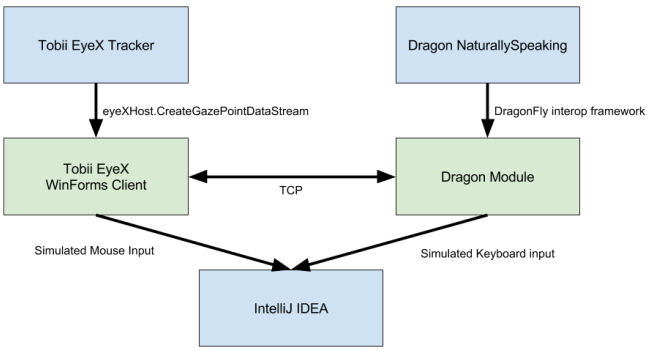

Figure 3: Overview of the software components of the prototype. Green denotes software implemented specifically for this prototype.

5.2.1 Tobii EyeX WinForms Client

The Tobii EyeX framework for C# enables the prototype to get a continuous stream of coordinates from the tracker bar. These coordinates are on the form (x, y), where both x and y is a **double** value of where the users eyes are looking at the display. These coordinates are calculated by taking the raw data from the tracker bar, and passing them through filters, attempting to remove outliers, and compensate for retinal jitter. This filtering is intended to get more stable coordinates, that hopefully reflect where the user is actually looking. The algorithm is proprietary, but is based on the work by Olsen [10].

The Tobii EyeX Winforms Client program supports 2 distinct commands:

- Jump to pane
- Go to gaze

These commands can be triggered either by using the registered hotkeys: $^1/_2$ (the key left of **1**, same position as $ on a Mac-keyboard) for Switch to pane, and **Ctrl** + $^1/_2$ for Go to gaze, or by issuing voice commands via the Dragon Module. These hotkeys were chosen, based on a variety of factors. It was important for us, to choose hotkeys that were

[9]IntelliJ IDEA https://www.jetbrains.com/idea/

[10]http://www.nuance.com/dragon/index.htm

not already used by IntelliJ IDEA, while at the same time choosing hotkeys that were not as akward as Ctrl + F1 had shown to be in the first evaluations. The $^1/_2$ key was chosen because it sits in a prominent position, and is almost never used in a software development context.

Our new implementation also does away with the delay that was noticeable in the first iteration. The first iteration switches panes by emulating keyboard shortcuts. Due to some limitations in IntelliJ shortcuts for pane switching, this introduced a slight, but noticeable delay. The new implementation removes this delay, because it does not use keyboard shortcuts for this functionality, but rather cursor emulation. To switch pane, it instantaneously moves the cursor to the correct pane, clicks and then moves the cursor back to the original position. This is meant to give the illusion that the pane where the user is looking grabs focus, without delay, or without moving the cursor from where the user expects it to be.

5.2.2 *Dragon Module*

The Dragon module consists of several parts. The commercial speech recognition product, Dragon NaturallySpeaking is used to recognize the users speech. Dragonfly[11], which is an open-source project written in python enables interoperability with Dragon NaturallySpeaking. This allows for the ability to execute custom actions, when the user speaks certain sequences of words.

When a regular command is received, the Dragon Module emulates keyboard shortcuts to achieve the desired result in IntelliJ.

An example of this is the *"create a for each loop"* command which creates a Java for-each loop. When the user speaks the voice command *"create for each loop"* (or a variant thereof) the Dragon Module emulates the keyboard input `fore`, and then presses Tab. IntelliJ then automatically expands `fore` to a for-each loop construct.

When a gaze-aware command is received, the Dragon Module sends a message via an internal TCP connection, to the Tobii EyeX Winforms Client. This message tells the Winforms Client to either switch to the pane the user is looking at, or to go to the point the user is looking at. After receiving confirmation that the Tobii EyeX Winforms Client has performed the task successfully, and the focus is at the correct place, the Dragon Module then emulates keyboard shortcuts to execute the command, exactly the same as in regular commands.

5.3 Evaluation

Based on the experiences with the evaluation of the first iteration, the evaluation methodology was changed for the final evaluation.

The assignment was modified from the assignment used during the first evaluation. The new assignment was still based on the exam sets from the Introduction to Programming course, but it was modified, so the user should complete it using Test Driven Development (TDD)[12]. The original assignment consisted of three Java classes, but one of them was used less than the other. This class was replaced by the class where the user should write the TDD tests. This was done to encourage more frequent switches between

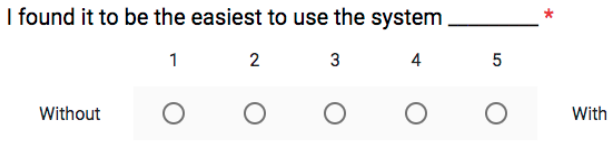

Figure 4: An example of a question in the questionnaire.

the three classes, and thus enabling the user to take better advantage of the capabilities the system offered.

During the evaluation of the first iteration, participants were asked to give feedback after each task, which had some drawbacks. The participants were able to give the highest score for one system, which meant that after evaluating the second system, they could not give it a higher score. When they used IntelliJ without extra modalities as the first product, they were asked to evaluate the IDE as a whole. This was not useful, as the user's thoughts on the IDE is not as relevant, as their comparison of the two sessions. Based on this, further inspiration was gathered [14][8], which resulted in a different approach for evaluating the second iteration. To make sure that the feedback based on the experience with the IDE could be compared to the experience with our prototype, participants were asked to do a combined evaluation after completing both tasks. Instead of rating the two products, they were this time asked to compare them to each other on a 1 to 5. As an example can be seen in Figure 4, where 1 would answer that the normal system was easiest, 3 if both performed equally and 5 if the prototype was the easiest to use. As in the evaluations in the first iteration, the questions are inspired by the set of computer usability satisfaction questionaires from IBM [8].

The participants completed the questionnaire after the task, then a small semi-structured interview was conducted on the basis of the answers they gave. The point of the questionnaire was never to use the scores in a quantitative way, but rather to use it as a tool for reflection, forcing the participants to consider all aspects of their experience. The participants were asked to elaborate on their scores after they had answered the questions, and it was these elaborations that were the most useful part of the evaluation.

Before the final evaluations a pilot study was performed, to allow for adjustment of the system and the evaluation procedure. After the pilot study, the final system was evaluated with three students, from the university campus. All of these users were currently following the Software Architecture course[13], which teaches TDD using Java and the IntelliJ IDE. All of the students were thus familiar with the TDD methodology and the IntelliJ IDE.

Each user evaluation was performed in the following order:

1. Introduction to the project
2. System calibration
3. Learning the system and the functionality
4. First task
5. Second task

[11]https://github.com/t4ngo/dragonfly

[12]https://en.wikipedia.org/wiki/Test-driven_development

[13]At the Department of Computer Science at Aarhus University - http://kursuskatalog.au.dk/en/course/64310

6. Questionnaire

7. Semi-structured interview

It should be noted, that for counterbalancing purposes, half of the participants still performed the first task without the added speech and gaze, and the second task without. For the other half of participants, this order was switched. Because there was an odd number of participants, 2/3 participants performed the first task with added speech and gaze. The setup can be seen in figure 5 and 6.

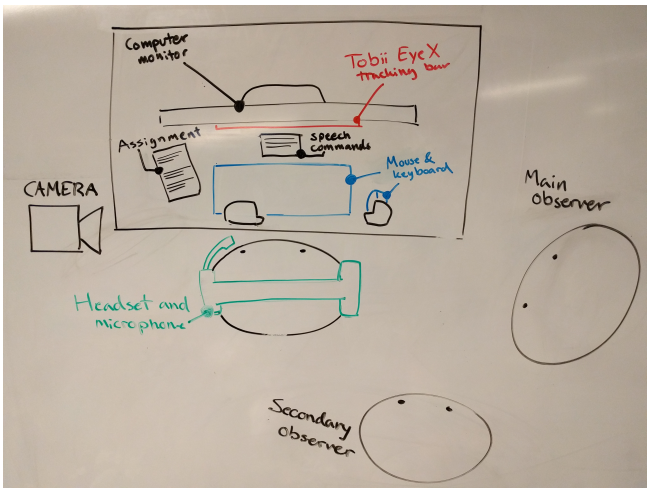

Figure 5: An illustration of the setup for evaluating the second iteration seen from above

Figure 6: Picture from the final evaluation. Note the camera is normally located at the pink circle, but is used to take this picture.

5.3.1 Pilot Evaluation

A pilot evaluation was performed to validate the design of the evaluation and to uncover any defects with the final prototype that might negatively influence the outcome of the evaluation. In order to achieve this, a user was chosen that didn't have previouse experience with the prototype. This particular user tried to complete the tasks with the prototype first, and then completed another set of tasks only using keyboard and mouse.

The user completed both tasks without serious issues, but the pilot study did uncover some usability-issues that were remedied before the actual evaluation. Most importantly it showed numerous occasions where the user couldn't remember the exact words to invoke a particular command, e.g. he might say "create method" instead of "new method", or attempt to run the test suite by saying a sequence of "tests. Run tests. Tests run", which would eventually make Dragon trigger the appropriate command. It was therefore decided to allow the user more freedom in how to express the commands. As an example, we made it so that all of the following commands triggered the creation of a *new method*.

1. New method
2. Create a new method
3. Create new method
4. Make a new method
5. Make new method

The intention of this was to allow the user to express the speech commands without necessarily needing to remember the exact command. Allowing for different formulations of commands, is also closer to the vision of the user simply thinking out loud what he wishes.

The subject also reported that he didn't feel confident in the speech recognition because the Dragon user interface visible on screen didn't provide adequate feedback when he spoke. This is something to be aware of, as time didn't allow us to design a better way to provide feedback. This is discussed in Section 6.

5.3.2 Final Evaluations

A lot of interesting findings were extracted from the final evaluations. Some of them were minor details concerning the interaction and chosen functionality, like the participants requesting the possibility of using speech-commands to not only create method but also field variables. Other findings were more about the overall experience while using the system and the potential of it. The participants first reactions to the system's capabilities were positive. Multiple participants exclaimed "Wow" when using the voice commands and the pane-selection for the first time, and they were clearly surprised by the system's response to their input. This aspect should be considered when reading the rest of the findings, as it indicates that the system has a novelty effect, that might skew the results towards the positive end of the spectrum.

5.3.3 Speech

The speech recognition was surprisingly robust against false positives, being turned on for the entirety of the evaluation, while the user performed the assignment, but also while thinking out loud, or exchanging mid-assignment remarks with the researchers. There was only one false positive, where the system thought the user gave a command, where he did not. The participant thought out loud *"don't I need a constructor here?"*, which the system (accidentally) reacted to, and presented him with a constructor-method where he was looking (and incidentally where it was needed).

In this case it was a mistake with a positive outcome, but in other circumstances could just as easily result in negative outcome. Some of the robustness can be attributed to the fact, that the thinking out loud and the conversation was in danish, while the commands were in english. There's a likelihood that there would be more false positives if these were both in the same language.

The accuracy rate of the speech recognition varied greatly from user to user. For one user, approximately 1/4 of the commands were not recognized correctly, while for other users, there was almost a hundred percent recognition rate. Some of the users also ran into trouble with the speech commands, because they had forgotten the correct command. One user tried saying *"create getter"* multiple times, but the system only responded to *"create getter**s**"*. The users generally recovered from these errors by saying the command again, with a different wording or a different tone of voice until it worked. In spite of these errors, when asked about errors in the post-task interview, they answered that they experienced no real errors, indicating that they had forgotten, or were not bothered by the incidents. It was expected that there would be errors, but not that the users would overlook them in the degree they did. They were however rather concerned about the aspect of speech-recognition in an open office environment. The participants expressed worry that the system would act on surrounding people's chatter and thus perform unwanted commands, or that a work environment where everyone was using voice commands would be unpleasant to work in.

A big problem with speech commands was the lack of feedback to the user. The speech-recognition is slow enough (appr. 0.5-1 second), that the participant didn't always understand whether or not the system had recognised their command correctly. A small window in the top showed what the system recognized, but this was small and hard to notice for the user.

5.3.4 Gaze

Most of the participants expressed that they felt much quicker when using the EGT to switch panes instead of the mouse. One participant said

> It's nice that I don't have to move my hand to the mouse anymore, now I can just press the button.

However the mouse was still used regularly, as the participants forgot they could use gaze, or needed fine precision work. The mouse was used to copy code from one place to another, or to navigate within a class. This is in alignment with the overall goal of not replacing existing modalities, but rather complimenting them.

The 'Go'-functionality where the user could move the caret to the gaze-position was too inaccurate to be useful. All participants tried it, but were confused and annoyed by it's inaccuracy. It was somewhat expected that the function would be inaccurate, but not that it would render the function completely useless in the way that it did. However this aligns well with previous work on the subject [1], [13], [20], saying that fine selection with EGT is not a viable option.

5.3.5 Overall

When provided with additional modalities, participants naturally used the modalities that were best suited for their current task. They would use the gaze-input to quickly switch focus to the desired editor pane. Then they would use the speech-commands to complete a more complex task like creating a new test. They would use the keyboard input to implement specific code, and the mouse to do precision pointing tasks eg. when copy-pasting code-segments. This is in agreement with Wang's work.[17]

Multiple participants expressed discomfort when they were to execute speech-commands that was gaze-dependant. They said it was a straining, that they had to keep their eyes focused on one spot for 1-2 seconds, while the speech-command was uttered and recognized. This aligns well with previous research, by e.g. Mateo et. al which note that gaze-based input can cause discomfort.[9]. It was also observed, that most of the time the participant would switch pane to the spot where they wanted the command executed, and then utter the command, even though the command was independent of where the caret was. The cause of this behavior is unknown, but we can speculate: Perhaps the participant didn't fully understand how the speech-command worked together with the EGT, or they felt that fixating upon one particular class was straining on the eyes. It is important to note however, that the system almost always executed the command in the correct context when using gaze-data, indicating that the users are focusing on the context where they expect the command to be executed. This supports the notion that gaze can be used to infer user attention. One participant suggested splitting the modalities so the speech-commands would be independent of the EGT and vice versa. This suggests that while the two modalities results in useful functionality separately, combining them in the way we have done is still suboptimal.

Error handling and feedback in the system still left room for improvement. If a user tried to perform an action that did not make sense (e.g. creating a test method in a non-test class, or generating getters in a class with no fields), the feedback the system gave was very poor to non-existent. When this happened, the researchers had to explain to the test subjects what had occurred. This is a direct consequence of emulating keyboard shortcuts for input, as there is no way of handling errors gracefully. A proposed solution to this is described in Section 6.

The evaluation suggests that the overall experience using the system is positive. Every participant rated the prototype with the added modalities higher in categories such as joy to use, and perceived effectiveness, no matter the order of tasks performed. Participants emphasized the speed of using eye movements and the fun and novelty of talking to he computer as factors influencing the pleasure of use. One participant said in regards to effectiveness:

> I felt more effective when using the prototype. I know that I was quicker when using the standard, but I had a better overview of my code when I used your system.

Another participant thought he was quicker and wrote code of higher quality when using our system compared to the standard. This is in line with two participants saying that they didn't feel they got more functionality when using our system, but rather more control, felt more productive and got a better overview of their code with our system.

It was clear that the participants hadn't gotten used to the system in the short time they had. It was observed multiple times that a participant would start writing code,

and then immediately deleting it to use the speech-command instead. This was most prominent during the last part of the task, which was more mentally tasking. This resulted in the participants focusing more on solving the task, and less on using the speech-commands, thus leading them to use their regular mouse- and keyboard work flows instead. The same thing was not observed to the same degree with the pane-switching functionality though, where all participants was quick to adopt the technique instead of using the mouse.

6. FUTURE WORK

The feedback from the evaluation shows that the proposed solution has potential, albeit it needs more work to be fully usable in a real world setting. Further studies into technology and the experience of using a system like this is also suggested.

As the system was only evaluated with 4 participants, it is recommended to do further evaluation with more participants from multiple backgrounds and skill levels. Furthermore, the evaluations were conducted using pre-determined tasks, and work-environments. Future work should aspire to make the system more robust and allowing it to be used under circumstances closer to real life use cases.

As described earlier, error handling in the system was never implemented, and thus there was no clear way for the user to recover from errors made by the system e.g. wrong speech-recognition or misread gaze-position. Further work needs to be done in this area. A proposal could be to get more data about the context by developing a plug-in for the IntelliJ IDE, thereby using info about the current code or editor pane to present the users with relevant options for error-recovery.

Both the researchers and the evaluation participants had concerns about the social awkwardness of users talking to their computer while others are listening, which would be a relevant research subject, if a system like this was ever to be implemented in an open office environment.

It is also recommended to examine how users will interact with he system over time. The evaluation suggested that some parts of the system took time to get used to, as the participants forgot to use some parts of the system as the tasks became more cognitively challenging. We suggest examining if this behaviour continues, as the gaze- and speech-modalities becomes a natural part the users' workflow. In combination with this, it would be interesting to study how easy the system is to incorporate into a user's current toolset, and how quickly the system will become ready-to-hand[14].

Multiple participants expressed that they thought the system could improve on ergonomic factors because they used the mouse less. It could be examined if the system reduces strain in the mouse-arm by using the gaze to point instead, as we assume it does. It is also possible that it will induce a greater strain on the eyes, which should be considered as well.

As the system is implemented now, most of the speech-commands are executed in short word-pairs e.g. 'new method' or 'create test'. It is worth researching, how the user experience would change if the interaction was through more natural language. E.g. 'I need getter methods in this class'. It is worth researching, if this would change the users perception or emotional attachment to the computer. Longer commands come with reduced efficiency however, and this trade-off would also need to be considered.

It would also be relevant to study the user experience of using a system where the feedback from the computer was auditive instead of visual. As described earlier, this conversational interaction[14] has been on the consumer markets for the past few years with the likes of Apple Siri and Amazon Alexa among others. But it could be examined how well it would fare in a context where the main focus is getting work done.

7. CONCLUSIONS

In this report we have presented the concept of using additional modalities to aid software developers in their daily work. With the purpose of evaluating the merit and qualities of this concept, a prototype that uses consumer grade gaze-tracking and speech-recognition was built and evaluated with computer scientist students in a controlled setting. We present the concept of gaze-aware voice commands, where the users gaze is used to disambiguate the voice commands of the user. Based on the evaluation we conclude that consumer grade gaze-tracking still isn't precise enough to be used as fine-pointing in a WIMP interface. It is however a viable option for switching focus to bigger objects like windows and areas within applications, which we demonstrate is useful for inferring user context and intent.

Using speech-commands for complex tasks was shown to be an aid to the users, as it removed the requirement of remembering arbitrary keyboard shortcuts or menu-tree structures. The participants expressed gratitude for the fact that they could just say their desired action out loud.

The participants in the evaluation were positive about switching editor pane using the gaze, and issuing commands using speech, but doing both to issue a command was less well received. We suspect that the combination of gaze and speech is not ideal because it combines a fast and a slow input-modality. Our gaze interaction was quick to execute (look, press, and response in less than 200 ms), while the speech-input was rather slow as it relied on heavy speech-recognition and complex input to the editor (between 500ms and 1500ms from input to feedback). This required the user to hold his gaze on the same position for some time, which was reported to be an unpleasant experience.

When provided with additional modalities, participants used the modalities that were best suited for their current task. They would use the gaze-input to quickly switch focus between screen areas, speech and keyboard in combination to perform more complicated actions, and mouse for precision navigation.

Every participant judged themselves to be more effective when having access to the extra modalities. Participants also rated their experience with the system as more pleasurable. From this we conclude that the addition of extra modalities to increase user bandwidth and make interactions more pleasurable is a promising avenue of research.

[14]A term originating from Martin Heidegger - https://en.wikipedia.org/wiki/Heideggerian_terminology#Ready-to-hand

8. ACKNOWLEDGMENTS

We would like to thank Marianne Graves[15], Niels Oluf Bouvin[16], Daniel Graungaard[17] for supervising and advising throughout the course of the project.

9. REFERENCES

[1] R. Bates and H. Istance. Zooming interfaces!: Enhancing the performance of eye controlled pointing devices. In *Proceedings of the Fifth International ACM Conference on Assistive Technologies*, Assets '02, pages 119–126, New York, NY, USA, 2002. ACM.

[2] R. A. Bolt. "put-that-there": Voice and gesture at the graphics interface. *SIGGRAPH Comput. Graph.*, 14(3):262–270, July 1980.

[3] P. R. Cohen, M. Johnston, D. McGee, S. Oviatt, J. Pittman, I. Smith, L. Chen, and J. Clow. Quickset: Multimodal interaction for distributed applications. In *Proceedings of the Fifth ACM International Conference on Multimedia*, MULTIMEDIA '97, pages 31–40, New York, NY, USA, 1997. ACM.

[4] A. M. David Rozado and D. Mazur. Voxvisio combining gaze and speech for accessible hci. *Resna 2016*, 2016.

[5] P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology*, 47(6):381, 1954.

[6] P. O. Kristensson. Next-generation text entry: Allowing users to flexibly combine speech, typing and gesturing. *Computer*, 48(7):84–87, 2015.

[7] M. Kumar, A. Paepcke, and T. Winograd. Eyepoint: Practical pointing and selection using gaze and keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 421–430, New York, NY, USA, 2007. ACM.

[8] J. R. Lewis. Ibm computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction*, 7(1):57–78, 1995.

[9] J. C. Mateo, J. San Agustin, and J. P. Hansen. Gaze beats mouse: Hands-free selection by combining gaze and emg. In *CHI '08 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '08, pages 3039–3044, New York, NY, USA, 2008. ACM.

[10] A. Olsen. The tobii i-vt fixation filter. *Tobii Technology*, 2012.

[11] S. Oviatt. Multimodal interactive maps: Designing for human performance. *Hum.-Comput. Interact.*, 12(1):93–129, Mar. 1997.

[12] S. Oviatt, A. DeAngeli, and K. Kuhn. Integration and synchronization of input modes during multimodal human-computer interaction. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, CHI '97, pages 415–422, New York, NY, USA, 1997. ACM.

[13] D. D. Salvucci and J. R. Anderson. Intelligent gaze-added interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '00, pages 273–280, New York, NY, USA, 2000. ACM.

[14] H. Sharp, Y. Rogers, and J. Preece. *Interaction Design: Beyond Human Computer Interaction*. John Wiley & Sons, 2007.

[15] J. Shaughnessy, E. Zechmeister, and J. Zechmeister. *Research Methods in Psychology*. McGraw-Hill Higher Education. McGraw-Hill, 2003.

[16] J. van der Kamp and V. Sundstedt. Gaze and voice controlled drawing. In *Proceedings of the 1st Conference on Novel Gaze-Controlled Applications*, NGCA '11, pages 9:1–9:8, New York, NY, USA, 2011. ACM.

[17] J. Wang. Integration of eye-gaze, voice and manual response in multimodal user interface. In *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, volume 5, pages 3938–3942 vol.5, Oct 1995.

[18] C. Ware and H. H. Mikaelian. An evaluation of an eye tracker as a device for computer input2. *SIGCHI Bull.*, 17(SI):183–188, May 1986.

[19] Q. Zhang, A. Imamiya, K. Go, and X. Mao. Overriding errors in a speech and gaze multimodal architecture. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*, IUI '04, pages 346–348, New York, NY, USA, 2004. ACM.

[20] X. Zhang and I. S. MacKenzie. Evaluating eye tracking with iso 9241 - part 9. In *Proceedings of the 12th International Conference on Human-computer Interaction: Intelligent Multimodal Interaction Environments*, HCI'07, pages 779–788, Berlin, Heidelberg, 2007. Springer-Verlag.

[15]Associate Professor, Department of Computer Science at Aarhus University, mgraves@cs.au.dk

[16]Associate Professor, Department of Computer Science at Aarhus University, bouvin@cs.au.dk

[17]Stud. cand.scient at Aarhus University, daniel@graungaard.com