Department of Electrical, Computer, and Software Engineering

Part IV Research Project

Literature Review and

Statement of Research Intent

Project Number:

How can Large Screens Help Programmers Understand Code?

Kevin Wu

Minghao Lin

Craig Sutherland

23/04/2023

Declaration of Originality

This report is my own unaided work and was not copied from nor written in collaboration with any other person.

Name: Kevin

ABSTRACT: Software development relies on a programmer's ability to comprehend code in different situations. When it comes to understanding code, developers tend to spend a considerable amount of time navigating and reading before they edit it. Many modern integrated development editors have come up with solutions to enhance program comprehension yet many of these interfaces have problematic designs like tabbing which limits navigation and spatial memory. Screen size is also a contributing factor to helping programmers understand code however many have yet to fully utilise large screens with specialised IDEs designed to help developers with positioning and mental mapping code. We will review in depth all research and studies related to these topics, and analysis common themes to explore how to combine these ideas and develop and implement a large screen based code editor.

1. Introduction

The study of how large screens impact coding comprehension has been an ongoing field focused on finding how different spatial situations affect a user's awareness when working around code. There have been many studies observing developer's interactions between different desktop screen environments including single compared to dual screen, and large displays compared to small displays. These studies remain an important part in finding ways to make programming more optimal for developers working in large code bases. Navigating around large source codes can take up most of their coding time hence reducing this will make coding more efficient and help improve coding comprehension. This literature review will also delve into how modern integrated development environments (IDEs) influence a programmer's ability to navigate and comprehend code. The overall goal of this paper is to see whether large screens can help programmers improve coding comprehension and form any wider discussions between these themes.

2. Related Work

2.1. Large Screens

A study [1] on the effects of screen sizes on the programmer's cognitive load found that dual screens gave the individual more spatial abilities to process and display code. This result was concluded by simulating a programming environment using projectors that displayed large projections of code and material. Data was collected via a pre-test before learning lessons with the projector in a controlled environment and once more after a four-week interval to see the cognitive load effects post evaluation.

Another study [2] found similar results where multiple desktops were set up to simulate a 2D virtual environment. The test conducted saw a single monitor against nine monitors (1,310,720 pixels versus 11,796,480 pixels) and found that users had improved performance navigating and completing visualization tasks in higher resolution. This can be contributed to more information and detail being shown at a higher resolution and thus decreased virtual navigation.

On the contrary, another study [3] found users preferred using medium sized displays over large displayed for performing non programming related tasks such as information combining, list searching, and colour matching. The results found that out of 18 participants, 12 preferred medium, 6 preferred large and 0 preferred the small display where the feedback for larger display was due to more head movement hence more physical demand and was easier to lose focus. Since each task was measured for task completion time (TCT) and error rate (ERR), the data showed the users performed faster using the smaller screen for certain tasks hence larger screen sizes did not correlate to faster performance rate. The TCT was dependent on the type of task allocated to the participant. This research however was conducted for non-programming purposes hence the negative response to large screen sizes may be considered irrelevant to the wider discussion.

A recent paper [4] observed code comprehension in an augmented reality environment. Participants of this experiment were given tutorials to become familiarised with HoloLens 2 (VR headset) and then tasked to fix bugs within virtual reality. By using VR, display size and space becomes limitless. The study found the developers who took part felt the tool improved code comprehension compared to using a traditional IDE and found the usability of the prototype as acceptable.

The only negative feedback received was the usefulness of resizing code panels and zooming which was rated below average. Not everyone agreed the tool itself was as efficient as traditional IDEs. One user states "In a normal IDE, the error would have been found more efficiently".

All summaries → need to think about what is common across all papers.

2.2. Integrated Development Editors and Software Tools

IDEs and other software tools help programmers develop and code more efficiently by combining features that increase productivity and code comprehension. By studying the relation between editors and how they improve coding comprehension, we can find if there are correlations between larger screens.

CodeRibbon [5] is one such user interface that helps developers by providing realistic development tasks for "workspace management" and "navigation". It is composed of grids of code for faster navigation between groups of code and allows dynamic movement between the ribbons. The grid allows juxtaposition that can move to display more code which overcomes the limitations of traditional IDEs where tabbed documents are standard interface configuration. This study on CodeRibbon found that Code Bubbles had faster navigation compared to traditional IDEs and code editors where 35% of the time was spent navigating the mechanics like tab switching and scrolling.

Code Bubbles [6] is also a user interface IDE that uses collections of code fragments grouped in sets to form working sets. The working sets consist of documentations, code methods and notes that can be repositioned across the interface screen via dragging. Unlike traditional IDEs where users navigate through tabs, Code Bubbles navigation is done through the user's spatial memory of where they placed the bubbles. A study used participants to perform tasks using Code Bubbles and give critical feedback. The tasks performed and observations were collected by the researchers and formed into quantitative data by filling out a questionnaire. The developer participants responded very positively, rating the interface 4.33 on a 5-point Likert scale based on convenience, 3.39 out of 5 for learnability, and 4.04 out of 5 for ease of use for reviewing code. Code Bubbles saw reduced operations by 54.64% compared to using Eclipse and 47.07% on average from other IDEs. The study does not the limitations, for one, it requires a lot more computer resources to operate including modern CPUs and hardware-accelerated graphics card, and importantly is not very effective on small displays

(requires either one large 24-inch monitor or two smaller 19-inch monitors to be consider effective.) It also lacks many features on regular IDEs including lack of support for other languages, lack of "quick fixes" and other similar features.

2.3 Code Comprehension and Spatial Relations

Code comprehension can be defined simply as how programmers gather information to understand code. To reach a high level of code comprehension requires an individual's spatial ability to quickly navigate through a code base.

This section is a bit jumbled

Results from the study [7] shows programmers using spatial positioning by spreading out paper with code and annotations helped the participants remember their positions, hence improving their spatial memory. The paper also found adding annotation helped programmers emphasize the paper as important for future use. A large-scale study [8] conducted in a professional development environment found that programmers spent 57.62% of their time on code comprehension activities, 23.96% on navigation, and 5.02% on editing code whilst the rest was spent on other tasks. This research emphasis the importance of reducing programming comprehension time to improve overall code comprehension. Another study [9] focused on code comprehension behavior, strategies, and tools used during software work environments. The research found that developers tended to follow "pragmatic comprehension strategies depending on context". This meant the participants preferred using solutions that were more informal like writing annotations, comments, commit messages to remember code compared to IDE features that supported better code comprehension. Another study [10] looked at sharing navigation data to improve program comprehension. Developers were given code to understand then had to complete basic tasks related to how the code functioned. A tool called Team Track was used by another group which showed code navigation patterns from fellow developers. The data showed improvement to code comprehension was subjective dependent on the environment although all developers agreed the tool had "merits for new developers". [4] offered unlimited spatial positioning for the user in augmented reality to read and annotate code.

3. Discussion

The research gathered from the studies of program comprehension shows a relation between large screens and code editors. In nearly all studies, the measurement of success for screen sizes and how effective an integrated development editor performed was based on the level of code comprehension by the developers and participants. These connections

provide useful insight in how this project may answer the missing links between a code editor designed for large screens. A relatively interesting pattern from the study of screens shows different ways large screens were interpreted. [2] depicts large screens based on pixel size whilst [3] is based on the diagonal length of the screen. Projectors were also used to display code when testing dual screens in [1]. A lot of these studies appear to be quite outdated hence may not be the most relevant in terms of the impact of screen sizes considering a single modern screen are much larger in size and in pixels hence when compared to two dated monitors from 2005. It is also important to note a single large display would show continuous pixels across the screen whereas a gap between two separate monitors or projections would pose a break of illusion to the developer. The study on VR programming was very fascinating as it showed limitless space for a display however the practicability of coding in an augmented reality is little to zero.

Another theme across all integrated development editors and software tools was the critical analysis of modern code editors like Visual Studio Code and Eclipse for using tabbing interfaces. The studies [5] [6] both mention tabbing interfaces leads to more time spent navigating when compared to newer IDE ideas like Code Bubbles and CodeRibbons. A common connection of navigation is seen across these studies and other code comprehension studies. The key focus for all new IDEs and tools was to reduce user navigation by providing more documentation on screen for the user to view thus reducing the need to swap between tabs. This plays a big part in this research as a larger screen would by coloration improve code comprehension by reducing user navigation since the user would have more space to view code and would mean the user is more likely to remember the spatial position of the documentations if they are not constantly changing tabs. Large screens also help with spatial memory since no zooming would be needed would make locating and navigating information quicker and require less cognitive load.

Based on these studies, we can observe a larger screen resulted in higher code comprehension for developers whilst code editors that focused on providing documentations ~~using~~ interfaces that allows ease of repositioning and movement also saw similar results. It is important to state that some studies for large screens were conducted for non-programming related tasks and for participants who had no software development experience hence may not be an accurate depiction of a developer's opinion towards large screens. Overall, these relations between the topics give insight into our research in creating a code editor that utilises spatial memory via large screens and how we can effectively conduct and gather quantitative data in studies related to users using IDEs and large screens.

4. Research Intent

Based on our literature review, we have reached an area of investigation. Our research aim is to find whether a code editor that utilises spatial positioning for large screens will help programmers understand a codebase better compared to a traditional tabbing integrated development editor. To help us seek answers to our question, we need to answer these sub-questions; will programmers utilise spatial memory similar to Craig Sutherland's [7] study around annotation and spatial positioning, will the use of our code editor be similar to traditional IDEs, will the effects of large screens impact the spatial memory of programmers and if true, to what degree, and what the limitations of traditional IDEs are when using large screens. To specify our study, large screens will be defined as electronic display panels of equal proportion to television diameter sizes.

We believe this project is worth doing as there is currently a lack of research towards television sized large screens and their impacts in code comprehension whereas there have been plenty on traditional monitor displays. Current existing IDEs like Code Bubbles and CodeRibbon do utilise spatial memory however there are no specific studies that specializes in large screen displays hence our project delves into connecting both topics and by building a code editor for large screen, we can find whether the hypothesis is true. Another reason this research may provide helpful is the current innovation and expansion of more affordable and larger displays to the average person hence one may begin to find them as alternatives to monitors in a software environment.

To accomplish this project, we will procced with additional research towards open-source material on creating an IDE. As of current, our code editor prototype will be built using the web-based Code Mirror component for React. Using our prototype, we plan on conducting a comparison observation between our code editor and a traditional tabbed IDE. Figure 1 shows a detailed timeline on our research project.

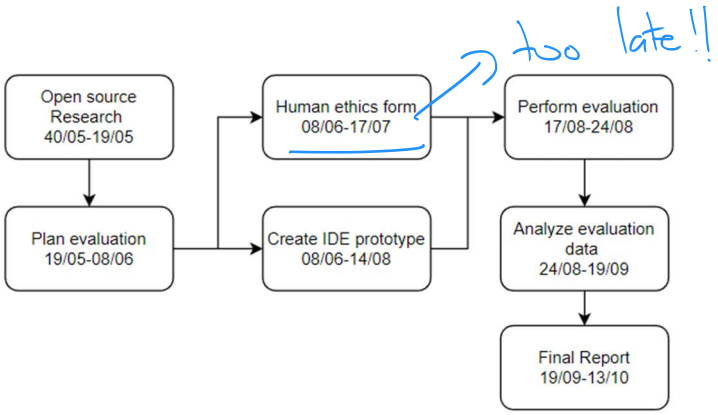

Figure 1 Project Timeline

Our observation/experiment will be conducted in the second semester and will be overseen by myself and my project partner. The experiment will see approximately (5-7) participants with programming experience use a traditional IDE with tabs on a large display then completing a questionnaire, then swapping over to our code editor and following the same procedure followed up with a short interview at the end. This will be conducted with lab access in the University of Auckland's Engineering building.

5. Conclusion

Upon close analysis of all studies related to large screens, alternative IDEs, and code comprehension, we can conclude that larger screens and IDEs that use spatial positioning do indeed improve programming comprehension for developers. Large screens with higher pixels helped programmers navigate faster by providing more information on display whilst IDEs like Code Bubbles and CodeRibbon broke off from traditional IDEs by allowing repositioning of documentations which showed improved navigation and more spatial positioning by participants. Based on this evidence, we have realized a gap in the research between code editors that ulitises large screens while also specialising in using spatial memory hence our research project will investigate this field. Our research project will help bridge this gap and find whether our code editor will help developers with large code bases more than IDEs like Eclipse or Visual Studios that uses tabbing for class navigation. This research will be valuable in furthering the study in finding more optimal ways for programmers to understand code and reduce unnecessary wasted time navigating and searching for information.

Acknowledgements

I would like to thank my partner Minghao Lin and our supervisor Craig Sutherland for providing the support and guidance to make our project successful.

References

[1] Chang, T., Hsu, J., & Yu, P. (2011). International Forum of Educational Technology & Society A Comparison of Single-and Dual-Screen Environment in Programming Language: Cognitive Loads and Learning Effectshttps://doi.org/10.2307/jeductechsoci.14.2.188

[2] Ball, R., & North, C. (2005). Effects of tiled high-resolution display on basic visualization and navigation tasks. ACM. https://doi.org/10.1145/1056808.1056875

[3] Klinke, H., Krieger, C., & Pickl, S. (2014). Analysis of the influence of screen size and resolution on work efficiency. Universität Stuttgart. https://doi.org/10.18419/opus-3489

[4] Diehl, S., & Weil, P. (Jan 1, 2022). IDEvelopAR: A Programming Interface to enhance Code Understanding in Augmented Reality. Paper presented at the https://doi.org/10.1109/VISSOFT55257.2022.00017 https://search.proquest.com/docview/2756566121

[5] Klein, B. P., & Henley, A. Z. (Sep 2021). CodeRibbon: More Efficient Workspace Management and Navigation for Mainstream Development Environments. Paper presented at the 604-608. https://doi.org/10.1109/ICSME52107.2021.00065 https://ieeexplore.ieee.org/document/9609170

[6] Bragdon, A., Reiss, S., Zeleznik, R., Karumuri, S., Cheung, W., Kaplan, J., Coleman, C., Adeputra, F., & LaViola, J., Joseph. (May 1, 2010). Code bubbles. Paper presented at the , 1 455-464. https://doi.org/10.1145/1806799.1806866 http://dl.acm.org/citation.cfm?id=1806866

[7] C. J. Sutherland, A. Luxton-Reilly and B. Plimmer. (2015). An Observational Study of How Experienced Programmers Annotate Program Code. Human-Computer Interaction - INTERACT 2015 (pp. 177-194). Springer International Publishing AG. https://doi.org/10.1007/978-3-319-22668-2_15

[8] University, Z., Bao, L., Lo, D., Xing, Z., Hassan, A. E., Xia, X. ;., Bao, Lingfeng, ;., Lo, D. ;., Xing, Z. ;., Hassan, A. E. ;., Li, S., & Measuring. (2018). Measuring program comprehension: A large-scale field study with Measuring program comprehension: A large-scale field study with professionals professionals Xin XIA Citation Citation

[9] Maalej, W., Tiarks, R., Roehm, T., & Koschke, R. (2014). On the Comprehension of Program Comprehension. ACM Transactions on Software Engineering and Methodology, 23(4), 1-37. https://doi.org/10.1145/2622669

[10]DeLine, R., Czerwinski, M., & Robertson, G. (2005). Easing program comprehension by sharing navigation data. Paper presented at the 241-248. https://doi.org/10.1109/VLHCC.2005.32 https://ieeexplore.ieee.org/document/1509509