This is a preliminary version of the paper that will appear in the
proceedings of AAATE 20017 published by ISO Press. DOI: 10.3233/978-1-61499-798-6-922

Accessibility Analysis of the Eclipse IDE for Users with Visual Impairment

Vanessa PETRAUSCH[1] and Claudia LOITSCH
Karlsruhe Institute of Technology, Karlsruhe, Germany

Abstract. Integrated Development Environments support software developers during their daily work. However, complex graphical interfaces and various functions disable an accessible development environment. We present the results of an online questionnaire, which identifies accessibility issues regarding the Eclipse IDE, and an inclusive tutorial for Eclipse for people with visual impairment.

Keywords. Accessibility, Eclipse, Survey, Visual Impairment

1. Introduction

Computer science has great potential for visual impaired people due to the text-based nature of programming. Advanced editors such as VIM [1], GNU Emacs [2] or complete audio-environments such as Emacspeak [3] are efficiently usable by blind software developers. However, the usage of such editors can be complicated when it comes to cooperative programming tasks. For large programming tasks, which often require several programmers to work together, an Integrated Development Environment (IDE) is preferred to simple text editors. IDEs reduce media disruptions by combining for instance, editors, compiler, interpreter, debugger, versioning system, formatting support and modelling editors into a whole. However, such integrated design results in complex interfaces including encapsulated menu structures, numerous views, and extensive functionalities, which can be confusing for users. Those visual-oriented user interface structures are even more challenging for blind programmers who cannot see spatial arrangements of interface elements and cannot use the mouse to operate the IDE. Nevertheless, many blind programmers use an IDE such as Android Studio, Visual Studio, or Eclipse.

Enabling visual impaired software developers to use comprehensive tooling requires accessible IDEs. That is, all features and tools must be compatible with assistive technologies such as screen readers, ensure contrast ratios, or provide well-documented keyboard shortcuts. The Eclipse IDE already supports good accessibility, for instance, by using the Microsoft Active Accessibility (MSAA) APIs to operate user interface elements with the keyboard and to ensure compatibility with screen readers [4]. Though, it is not sufficiently analysed how accessible integrated programming tools such as

[1] Corresponding author, Study Centre for the Visually Impaired Students (SZS), Karlsruhe Institute of Technology (KIT), Engesserstr. 4, 76131 Karlsruhe, Germany, E-Mail: vanessa.petrausch@kit.edu

navigating source code, debugging, or code folding are. Investigating end-user feedback relating to specialized IDE functionalities is at the core of this work.

The remainder of this paper is structured as follows: Section 2 discusses related work, section 3 describes the methodology of the questionnaire and section 4 presents the results and discusses the findings of the questionnaire. Section 5 describes further processing of the results and section 6 concludes the paper and provides an overview of future work.

2. Related Work

Only few similar studies were conducted to research the needs and strategies of blind software developers. In a study conducted by Mealin and Murphy-Hill [5], general working strategies of blind software developers were investigated. They show that blind developers often perform so-called "out-of-context editing" when they seek certain aspects in the source code. Out-of-context editing – contrasting to editing in-place – means that source code is copied over into another editor or into an empty text file which makes it easier to navigate through the source code, for instance, by jumping to the beginning and to the end. Though, the study did not focus on the accessibility of certain IDEs such as Eclipse. Only general programming strategies of blind developers were investigated. Albusays and Ludi performed a survey with 69 blind developers and they found that navigating large source code sequentially (by using arrow keys) is one of the most prominent challenges of blind developers [6]. Their results support the findings of Mealing and Murphy-Hill [5] that some blind developers use external editors to better navigate and not to lose orientation.

Other papers focused on related educational aspects of visual impaired computer scientists. For instance, support for learning Java by addressing special needs of people with visual impairments was addressed by JavaSpeak [7], which is an integrated IDE with aural feedback and keyboard navigation control. In its current version 3.0, the functionality has been implemented as a set of plugins for Eclipse. However, the accessibility of common programming features, such as auto-completion, jumping to the declaration of methods, viewing source code documentations, or re-factoring, were not addressed. The requirements of visual impaired students when programming Lego Mindstorms NXT robots was investigated by Ludi et.al [8]. They conducted an experiment in which they tested the accessibility of JBrick and found that navigating code is not ideally supported and could be improved by using audio cues.

3. Method for investigating the accessibility of Eclipse

An online questionnaire was conducted with 14 blind or partially sighted persons to gain insights into programming workflows of visual impaired people. We were particularly interested in the Eclipse IDE because it already provides good accessibility support and it is known that many blind programmers use Eclipse as their programming environment. The questions were elaborated within an interdisciplinary group of persons consisting of sighted and blind programmers as well as accessibility experts without programming knowledge Thus, our questionnaire addresses many topics ranging from general programming experiences to questions regarding accessibility experiences with screenreaders and Braille displays. We also included questions about common problems,

experienced by blind programmers during their daily work, to reveal different solution strategies. Our blind colleague tested the accessibility of the online form beforehand. The questionnaire was available in German and English and open for every interested person. We acquired participants by personal mail. Moreover, we requested all participants to forward the invitation. We also published the announcement in forums for the blind. The questionnaire was organized into three categories with 33 questions in total. In the first category, we asked for personal background. The second category of questions treated programming experiences in general and the third category was related to specific programming tasks and their accessibility in Eclipse.

The first part included four questions about age, job status and area, and the kind of visual impairment. The second part started with basic questions about programming skills and used environment. For instance, we asked how often the participants use Eclipse, which version they use, and how long they have been into the field of programming. The third part included more specific questions about concrete programming scenarios. For instance, we wanted to know if and how errors and warnings are ascertained, how participants keep track of source code changes (e.g. by using versioning control systems), if they know and use code folding, and how they navigate through source code. It also covered special accessibility topics of Eclipse such as how well it can be used in conjunction with a screen reader or a braille display. The last two questions of the questionnaire allowed the participants to write free text as the answer. First, we asked for features the participants would like to have to be included in Eclipse. Second, we asked for further comments the participants would like to give.

4. Results of the questionnaire

This section summarizes the results of the questionnaire along our three mentioned categories. First, we describe the participants of the questionnaire and we summarize the answers from the participants according to their general usage of the Eclipse IDE. Afterwards, results concerning specific programming tasks and tools are presented. The last part summarizes the accessibility of Eclipse using a screen reader or braille display. The complete results of the questionnaire can be downloaded from our webpage[2].

4.1. Participants and general usage of Eclipse

In total, 14 participants between 18 and 60 concluded the questionnaire. The majority (58%) was aged between 30 and 45, 45.3% between 19 and 29, and one participant (8.3%) was aged between 46 and 60. 66% of the participants were from the ICT domain and 25% from universities. The majority of the participants (83%) was concerned with programming tasks for more than five years.

Eclipse was used as the IDE with the exception of one participant. The frequency how often Eclipse was used by the participants is summarized in **Table 1**.

At the time of our study, 50% of the participants used Eclipse Version 4.5 (Mars). Luna (7.1%), Kepler (7.1%), Juno (7.1%), or other versions (12.3%) were also used. Visual Studio was mentioned as an alternative to Eclipse by 28.6% of the participants.

[2] http://www.cooperate-project.de/images/publications/EclipseSurvey.pdf

Table 1. Frequency of using Eclipse

Frequency	Count in %
Daily	16.7
Several times per week	33.3
Several times per month	16.7
Seldom	25.0
Never	8.3

4.2. Qualitative evaluation of general programming techniques

Primarily, we were interested in the approach of *recognizing errors and warnings* within the code, since this is a known problem for visually impaired programmers. From a visual perspective, errors and warnings are usually expresses by icons. A user can hove the mouse over the icon to get insights how to fix the problem as shown in **Figure 1**. That is, errors and warnings can be found quickly by sighted developers but not by blind developers.

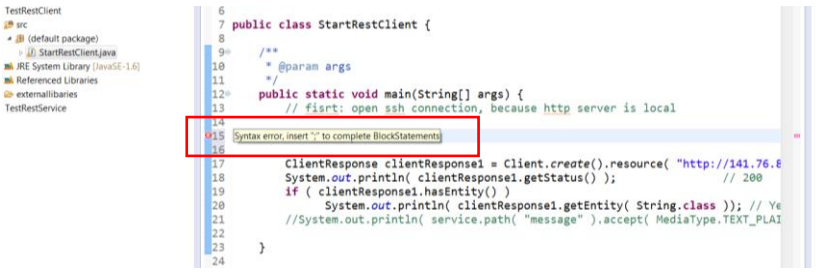

Figure 1. Example of visual representation of errors in Eclipse

Table 2 lists the approaches how the participants recognize errors and how they get detailed information of perceived errors.

Table 2. Approach to recognize errors and warnings non-visually.

Possible Answers	Count in %
Visual Differentiation	16.7
Read aloud by screen reader	41.7
Keyboard Shortcut	25.0
Braille Display	16.7

The awareness of accessibility features and strategies to recognize errors and warnings varies between the participants. For instance, one participant did not know that some screen readers provide functionalities to announce errors. On the other hand, using a shortcut to jump to the list of errors with CTRL+F7 or circling through errors and warnings with CTRL+POINT/CTRL+COMMA is the most frequently used approach and is assessed as a sufficient solution by the majority of the participants. However, it was noted that there is no real-time support in recognizing errors while writing. Moreover, it was remarked three times that recognizing errors on the braille display would be a useful option, but it seems that this is not working properly with current versions of the screen readers JAWS and NVDA. Additionally, 78.6% of the participants answered that supporting acoustic messages (e.g. earcons) would be helpful to announce and distinguish between errors and warnings. However, 28.6% said this kind of support

would be only partially helpful, though, they did not give any hints in which situations it would not be helpful. We received meaningful remarks (4 comments were given) that incorporating auditory feedback is crucial. For instance, one user gave the following remark:

"I think sounds are effective for alerting users to events, errors or notifications. That helps me to switch my focus to the event / error in question. But auditory languages are harder to design; one of the options in Jaws was to have control types announced by sounds rather than speech, but I remember it was hard for me to find the right metaphors for each control and remembering all selections. So I think an effective auditory language should be simple, with a small number of sounds and the sounds should be a clear metaphor for the event they represent."

Another user mentioned that they give preferential treatment to auditory messages over speech output. A short, clear distinguishable tone is favoured to perceive additional feedback such as errors and warnings, in particular, when speech output is used at the same time to read aloud source code, comments, or inputs. However, it has been stated that auditory messages must be configurable in terms of volume and it must be possible to deactivate auditory cues. Moreover, it was mentioned that the duration and quality of the tone is crucial. For instance, one user stated: *"It is important that an auditory message is below 0.5 seconds and that echoed recordings are avoided".* 21.4% of the participants reject additional acoustic feedback.

The third specific accessibility aspects that was surveyed regard the *experiences in handling code changes of other developers* (e.g. by using version control system). The general approach used by the participants at the time of the questionnaire is summarized in **Table 3**.

Table 3. Approach to follow source code changes by other developers

Possible Answers	Count in %
Integrated version control	14.3
External version control	42.9
Manual code comparison	14.3
Personal demand	7.1
Others	14.3
No answer	7.1

The majority of participants (42.9%) stated to use external version control systems. The reason why these participants do not use an integrated version control system is limited accessibility. Besides inaccessibility, one participant explicitly stated, external version control systems (e.g. command-line interfaces) are the preferred way to keep all different projects synchronized. Participants, who use an integrated version control system of Eclipse (14.3%), were further asked which Eclipse feature they use (side-by-side diff or change history). All participants (14.3%) use the change history. The majority of the participants (57.1%) gave additionally feedback with respect to the accessibility of versioning control. In summary, displaying complex changes (21.4%), traceability of the author (7.1) and identification of the changed components (28.6%) were the most challenging aspects in terms of version control.

The forth surveyed aspect dealt with *code folding*, which is a feature to selectively hide and display sections of a source code file. Half of the participants (50%) use code folding, 35% do not use this feature, one participant (7.1%) did not know the feature, and again one participant (7.1%) did not reply. The reason why code folding is not used range

from the inability to identify hidden areas, missing or unknown shortcuts, and the lack of clarity about the usage of this feature.

The fifth surveyed aspects were about the strategies to *navigate through source code*. Half of the participants (50%) navigate within the source code file itself. Only one participant (7.1%) use the outline view, which is a special feature that facilitates navigating complex source files. Two participants (14.3%) make use of the text search to find and jump to certain places (e.g. function, variable) in the source code. Four participants (28.6) stated to combine several approaches.

4.3. Qualitative evaluation of accessibility issues

The last part focuses on using Eclipse with the help of a screen reader or a braille display. **Table 4** summaries the usage of a screen reader and a braille display. Only one person stated not to use a screen reader at all. Other participants mainly use screen reader at a windows system (NVDA and Jaws) and two use a Linux screen reader (ORCA). Additionally, 72% of the participants additionally use a braille display - three do not use any braille display and one did not answer the question.

Table 4. Summary of screenreader and braille usage

Possible Answers	Count in %
NVDA	50.0
Jaws	28.6
ORCA	14.3
Braille display	71.4
No Braille display	21.4

Although many participants use a screen reader, 44% of them stated that the usage is only moderate and 21% answered that the usage is good or very good. Despite the good ranking, many participants reported on problems with screen readers. 65% reported many different problems, such as the inaccessibility of 3rd party plugins, performance problems, hanging cursor, incomplete access to GUI elements, and the absence of announcing real-time information.

Moreover, there is a consensus that the participants did not make any changes in the Eclipse environment to enhance the accessibility. Only few participants (28%) did minor adjustments of the view arrangement and key bindings. One participant only installed the accessibility plugin ACTF.

To sum up, the questionnaire shows that visual impaired developers most commonly use Eclipse and it provides already good accessibility. However, the results also indicate that major improvements can be achieved by making errors and notifications more accessible, for instance, by providing acoustic feedback. However, it must be ensured that auditory cues are not the only way to ascertain errors. That is why some participants reject using audio feedback but also because representing errors solely by using audio messages would again decrease the accessibility for other target users, for instance, people with hearing impairments. Presenting errors and warnings in alternative formats (visual, auditory, tactile) is the recommended approach here.

Moreover, the results of the questionnaire show that specialized features (e.g. code folding) or views (e.g. outline view), which shall simplify programming tasks, are used rather seldom by visual impaired developers due to complexity of those features, missing awareness and accessibility issues. Code folding could be a very useful feature for visual

impaired developers to overcome barriers when navigating large source code. However, how code folding is currently designed and implemented only addresses the needs of sighted people. Intuitive ways to keep track of and orientation in collapsed regions as well as support for corresponding shortcuts and feedback must be still designed and evaluated.

The results also expound accessibility issues with version control systems – it cannot be stated that these issues affect only integrated version control systems and do not occur in external version control systems as well.

Finally, several participants asked for an accessible tutorial of Eclipse at the end of the questionnaire. The tutorial shall improve the understanding of the view and perspective concept of the IDE. Equally, it is very likely that good documentations of the Eclipse functionalities are going to improve and facilitate the efficiency when using the IDE.

5. Further processing of the results

The analysis of the results of the questionnaire inspired us to create two documents to improve the usage of Eclipse for blind persons. On the one hand, we designed a tutorial for people with visual impairment to facilitate the introduction to Eclipse. It focusses on instructions and tips for the practical usage of Eclipse. The introduction includes necessary accessibility configurations to provide an effective workflow with Eclipse. It also describes two short examples for a programming workflow: importing an existing project and creating a new one. Shortcuts are provided in each section to enhance working techniques and are also summarized at the end to provide a quick reference. To allow this the shortcuts were grouped by tasks and included a textual description and the shortcuts in a tabular way as can be seen in **Table 5**. One section especially focusses on how to handle known problems with Eclipse, which were found by the analysis of the questionnaire. The tutorial closes with an overview of commonly used shortcuts. The German version of the tutorial is available on our homepage[3].

Table 5. Excerpt of the quick reference list of shortcuts

General Navigation Elements	
List all shortcuts in Eclipse	CTRL+SHIFT+L
Cycle views	CTRL+F7
Open View (even not displayed ones)	ALT+SHIFT+Q
Set focus on menu	F10
Set focus on editor	F12
Programming	
Show Outline	CTRL+O
Go to declaration of selected element	F3
Cycle errors forward/backward	CTRL+Point/Comma
Show quick fix, navigate options with arrow keys	CTRL+1
Rename selected element and all references	ALT+SHIFT+R
Go to next method	CTRL+SHIFT+Arrow up/down

In the long term, we will create guidelines for developers of IDEs which should raise the awareness of accessibility during developing tasks. It should support developers to include accessibility features during their development process, e.g. including all GUI

[3] http://www.cooperate-project.de/index.php/en/downloadmenuen/training-materials

elements in the tabbing order or to provide meaningful naming of buttons. People with visual impairment rely on meaningful naming because otherwise they only get the information that there are buttons, but not which button provides which functionality. Hence, the guidelines will include an overview of different accessibility issues and will provide practical tips to develop accessible software.

6. Conclusion

The accessibility status of the actual Eclipse IDE was investigated by using an online questionnaire. The questions focused on general information of programming techniques of blind persons and especially relating accessibility issues in this context. 14 participants with visual impairment completed the questionnaire. The results of the questionnaire and particularly the comprehensive comments from the participants provide a good overview of accessibility issues regarding the Eclipse IDE and point at further improvement of its' accessibility. The results already led to the development of an inclusive tutorial for the Eclipse IDE. In the future, the results will help us to develop general guidelines for accessible software development. We expect that these results will raise the awareness of accessibility within software development and facilitate the development process of accessible software.

References

[1] B. Moolenaar, "The Vim Editor," 2008. [Online]. Available: http://www.vim.org/. [Accessed 30 January 2017].

[2] R. M. Stallman, Gnu Emacs Manual: For Version 22, Free Software Foundation, 2007.

[3] T. Raman, "Emacspeak - a speech interface," *Proceedings of the SIGCHI conference on Human factors in computing systems,* pp. 66-71, 1996.

[4] Eclipse Foundation, "Eclipse IDE User Guide," [Online]. Available: http://help.eclipse.org/neon/index.jsp?nav=%2F0. [Accessed 30 January 2017].

[5] S. Mealin and E. Murphy-Hill, "An exploratory study of blind software developers," *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on,* pp. 71-74, 2012.

[6] K. Albusays and S. Ludi, "Eliciting programming challenges faced by developers with visual impairments: exploratory study," *Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering,* pp. 82-85, 2016.

[7] J. M. Francioni and A. C. Smith, "Computer science accessibility for students with visual disabilities," *ACM SIGCSE Bulletin,* pp. 91-95, 2002.

[8] S. Ludi, L. Ellis and S. Jordan, "An accessible robotics programming environment for visually impaired users," *Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility,* pp. 237-238, 2014.

[9] A. King, P. Blenkhorn, D. Crombie, S. Dijkstra, G. Evans and J. Wood, "Presenting UML software engineering diagrams to blind people," *International Conference on Computers for Handicapped Person,* pp. 522-529, 2004.