Visual debuggers and the deaf: paving the way to workplace

Marcos Devaner do Nascimento[1], **Francisco Carlos de Mattos Brito Oliveira**[1], **Adriano Tavares de Freitas**[2], **Lidiane Castro Silva**[1]

[1]Computer Science Department – State University of Ceará (UECE)
Itaperi Campus, Fortaleza – CE – Brazil

[2]Computing Department – Federal Institute of Ceará (IFCE)
Maracanaú Campus, Maracanaú – CE – Brazil

marcos@projetolead.com.br, fran.mb.oliveira@gmail.com

{tfreitas.adriano, lidcastro}@gmail.com

Abstract. *It is not enough to offer accessible learning material, accessible learning platform and bilingual tutoring. Deaf or hearing impaired (DHI) java graduates still show poorer perfomance in debugging tasks when compared to their hearing counterparts. Direct manipulation and visual debuggers might improve the odds of securing the DHI a position in the IT industry. We present a study where DHI java graduates perform debugging tasks using a visual debugger and Eclipse. Situated analysis, usability assessment and a formal task performance evaluation show visual debuggers might benefit the DHI programmer.*

1. Introduction

Brazil has 9.7 million people with a hearing disability (PWD) [Census 2010]. Although there are government projects with the aim of promoting educational and social inclusion of this public, the educational process, the professional field and world knowledge are deficient [Santiago 2011]. Thus, it is hard for deaf people to achieve competitiveness in the labor market.

The young deaf prefer courses in administration (56%) or in technology (31%) [Santiago 2011]. Jobs in technology present an opportunity to improve their lives since there are many vacancies and training time is relatively short. Moreover distance education is a solution for the PWD in remote areas or with mobility difficulties.

The *Laboratory of Distance Education for People with Disabilities* (LE@D lab) creates and offers, through our accessible learning management system (LMS), seven IT related courses, encompassing more than 900 of training hours. Although our LMS is equipped for the DHI and those with missing limbs, the focus of this text is on the DHI java graduate.

Amonng our efforts to create more learing opportunities and keep the DHI motivated, in our LMS, Java programming workshops are held on a environment that allows the collaboration of a tutor, a translator and the DHI [Silva et al. 2014].

We've seen that it is not enough to train in order to secure a position in the workplace. Our graduates still struggle to secure a position in IT industry. This is especially

DOI: 10.5753/cbie.sbie.2015.792

true for the DHI. There is the fear they will not match the performance levels of the hearing counterpart. Therefore the challenge extends to that of task analysis and design.

We are interested in empowering the DHI programmer in the daily tasks of a regular software engineer, such as software evolution, debugging. We have learned that DHI graduates from our courses had inferior performance in debugging tasks when compared to hearing counterparts who took the very same courses [do Nascimento et al. 2014].

Visual debuggers might represent hope for improving the performance of the DHI programmer. In this paper,we compare how a visual debugger (JGrasp) impacts the activities of a DHI programmer. Ten participants were recruited to debug code in Eclipse and JGrasp, in a between-subjects design. In that study, all subjects used industry-standard Eclipse programming environment.

Performance was measured by: 1) Time to complete the task (TCT); 2) Number of times the subject asked for external help assistance (HA) and 3) Number of tasks completed successfully (TCS). Based on the *unpaired t-test*, the results point to JGrasp as a more productive tool. At $p \leq 0.10$, the results are not statistically significant concerning to the TCT and HA values, where we get *p-values* of 0.17 and 0.13 respectively. However, we observe significant difference in the TCS variable with a *p-value* of 0.08. The TCT and TCS values point advantage for JGrasp, the participants were able to finish more tasks demanding less time. A questionnaire based on the System Usability Scale (SUS) [Brooke 1996] was applied. The average SUS score for JGrasp was 72 and 50 for Eclipse. At $p \leq 0.05$, the *unpaired t-test* give us a *p-value* of 0.01, thus we can conclude that JGrasp has also a better usability.

The structure of this paper is as follows: we present the theoretical background in Section 3, which deals with direct manipulation and development environments. In Section 4, we present related work concerning visual debuggers. Section 5 shows our study design and subject profile; and in Section 6 we present and discuss the results. Finally we present our proposal to build an integrated development environment (IDE) that should keep the gains of an accessible java learning environment, as reported in [Silva et al. 2014] and add a direct-manipulation inspired visual debugger.

2. Previous Work

2.1. JLoad - A Java Learning Object to Assist the Deaf

The *JLoad* [Silva et al. 2014] shown in Figure 1 was made to teach the first notions of Java to junior students. It enables Java course designers to create their own workshops and include them into the course's webpages. A workshop in the *JLoad* is comprised of a list of programming activities, and each activity has its own set of instructions (displayed in portuguese and Libras). Typically, an activity in the *JLoad* requires some coding, compiling and running a Java program. The student can submit the code for grading, asking and getting situated help from a tutor - all done within the environment of the *JLoad*. This all-in-one approach prevents the learner to install and learn how to use an integrated development environment such as Eclipse. The student can send the tutor inquiries about a workshop's task. The student can even highlight the portion of code she is in doubt with. The highlighted code is anchored to the chat and vice-versa.

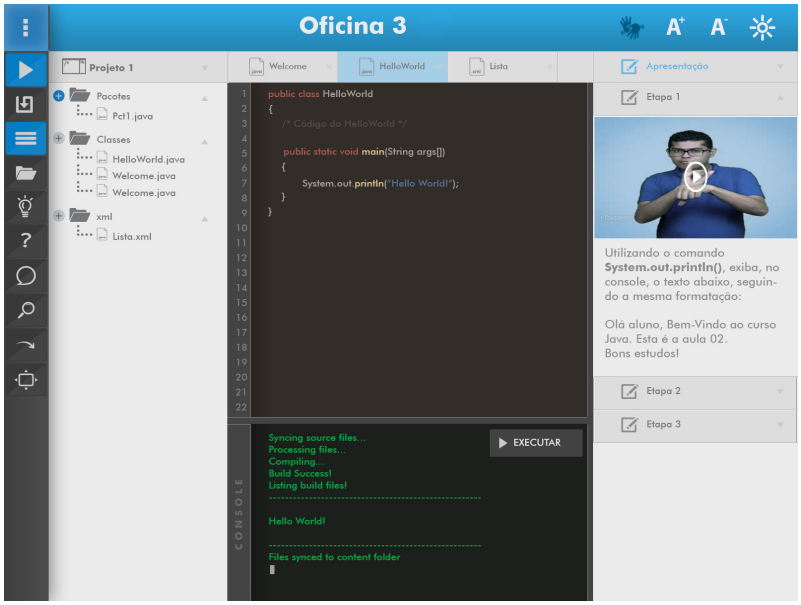

Figure 1. JLoad Interface.

2.2. Debugging a java code: the perfomance of hearing impaired programmers

The performance of deaf and hearing impaired (DHI) and Non-DHI programmers when debugging Java code was assessed in [do Nascimento et al. 2014]. Ten participants (five DHI and five Non-DHI) used the Eclipse IDE's debugger to perform some depuration tasks. The main idea of the experiment is to compare the performace of both groups. Basically, we measured the time taken to complete the task, the number of requests for assistance and successfully completed tasks.

We can see from the results (Table 1) that DHI participants underperformed the listeners, taking into account the variables of time, the success and aid in the task. In a qualitative analysis, certified, by the testimonies of the DHI participants that they found it difficult to understand the messages passed by the system and interpretation of icons arranged.

The work shows that Non-DHI and DHI had significative differences in performance. There is evidence of correlation between the hearing condition and the ability to complete tasks related to Java debugging in the Eclipse – $\chi^2(1, N = 70) = 17.43$, $p < 0.001$. There is also evidence that DHI take long to complete (when they do complete) the tasks – $t(44) = 2.54$, $p = 0.0153$.

3. Direct Manipulation and IDE's

According to [Hutchins et al. 1985], direct manipulation are visual interfaces in which users operate on a representation of objects of interest. For [Rose et al. 1995], using the visibility of objects and actions on these objects produce a significant difference in productivity in interaction with systems. In some studies, the interfaces with these features allow the user a better domain of the interface; better competence in performing tasks; ease of learning both basic functions as advanced; confidence that will continue to dominate the interface even if they stop to use it for a while; satisfaction in using the system; will of teaching others; and desire to explore more advanced aspects of the system

Table 1. Results of the Perfomance Experiment.

	Mean Time to complete (mm:ss)		# requests for help		completed the task	
	Non-DHI	DHI	Non-DHI	DHI	Non-DHI	DHI
Task 01	00:31	03:59	0	2	5	5
Task 02	19:38	17:37	3	1	4	1
Task 03	02:43	15:59	1	1	5	1
Task 04	03:39	-	2	0	3	0
Task 05	00:50	22:45	0	3	2	4
Task 06	03:15	-	2	0	5	0
Task 07	01:29	06:52	2	4	4	5

[Rose et al. 1995].

Some integrated development environments (IDEs) have applied not only the concept of direct manipulation as well as the concept of visualization, doing complex tasks like debugging a code simpler and more intuitive. [Moreno and Joy 2007] say that because of the ease of using these instruments, which are intended for the early stages of a programmed learning process, the strategy is to make the objects and values visible and manipulated graphically.

[Cypher and Halbert 1993] show that the concepts of direct manipulation applied to development environments can generate greater productivity for developers. It is a concept that helps in the development of logic, since materializes something abstract, allowing less cognitive demand for the interpretation of mathematical logic and also allowing the implementation of computational logic in the development of algorithms.

DHI are very visual [Gesueli and de Moura 2006] and it is reasonable to assume that they will benefit from an IDE with these characteristics .

4. Related Work

Visual debuggers implement, by its conception, direct manipulation strategies. In this section, we present several visual debuggers, briefly discuss how they implement such strategies and how benefits they brought to pupils and programmers. We also choose one of them to use in the study reported in this text and justify our choice.

4.1. Visualizing Programs with Jeliot 3

Jeliot 3 [Moreno and Joy 2007] is a tool designed for pupils to learn procedural or object-oriented programming. Its main feature is the total or partial view of source codes and control flows. Using this tool,students can develop and, at the same time, see the visual representation of a running code. During this process students acquire a mental model of computing that helps to understand better the construction of the program.

The system is easy to use. It has consistent, complete and continuous view, supports viewing a large subset of programs written in Java. The layout of this tool is divided into four parts: methods, constants, area for expression evaluation and instance area as we can see in Figure 2.

This program seeks to be as consistent as possible in order to reduce the cognitive

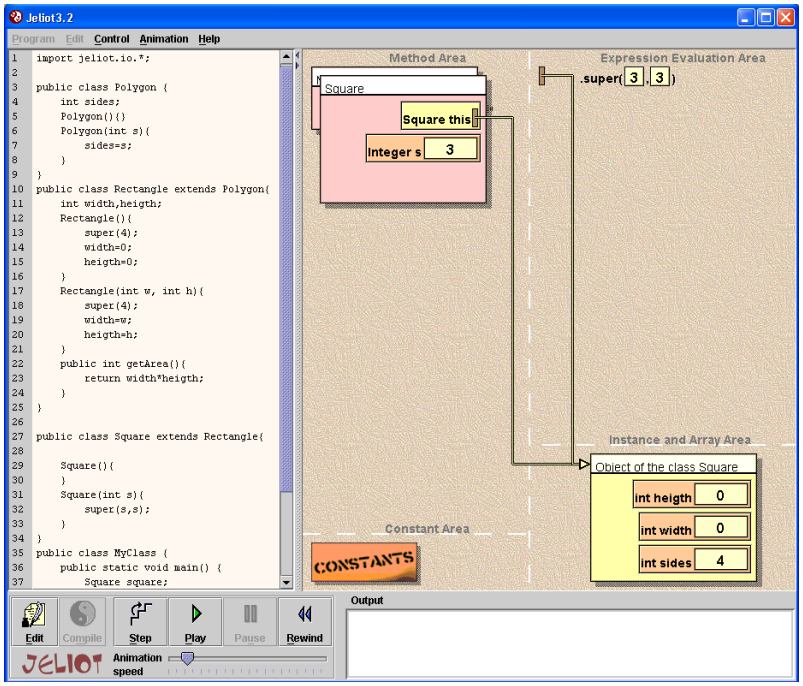

Figure 2. Jeliot Interface

load of students during learning. Although this tool displays a debug of visual objects, in a critical evaluation we realized that it has a series representations and diagrams that could confuse the user. Complex diagrams easily become confused and difficult to read. Studies show that graphical approaches are more efficient when the task requires pattern recognition, but not when the visual field is too full of objects and the task requires detailed information [Graciano 2007].

4.2. Java Interactive Visualization Environment (Jive)

Jive [Cattaneo et al. 2004] is an interactive execution tool developed by the Department of Science and Computer Engineering from the University of Buffalo. This system is used for: 1) debugging Java programs with rich views of object structure and interactions between methods; 2) facilitating maintenance software; providing insight into the dynamic behavior of programs and 3) teaching and learning Java.

It was originally designed as a stand-alone Java application. It has recently been redesigned to the Eclipse platform and consists of a set of plug-ins and features. Its distribution takes place using the Eclipse update manager. It provides two main views to display the running Java programs: the object diagram view and the sequence diagram view.

This tool uses the object diagram that demand prior knowledge of this type of diagram for better understanding of what is being presented, which can generate DHI greater cognitive effort by removing the focus of logic to the concepts linked to the diagram (Figure 3).

4.3. JGrasp

JGrasp [Cross et al. 2004] (Figure 4) is an IDE developed to provide dynamic and illustrative views of Java data structures. These views are generated automatically and syn-

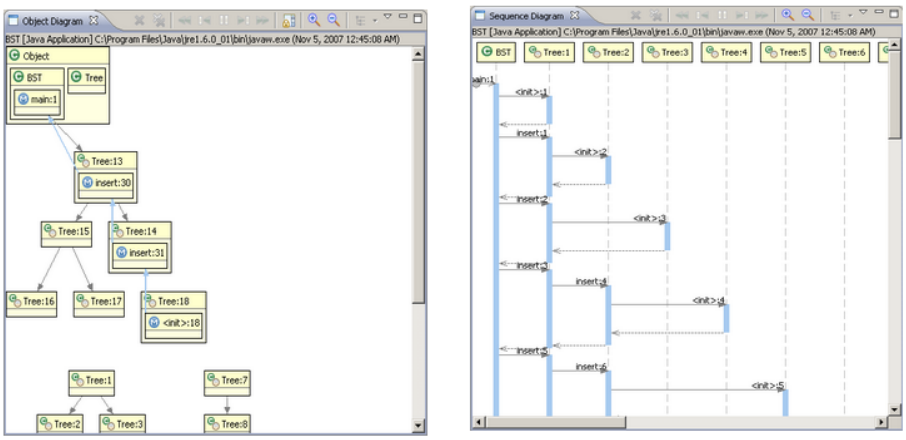

Figure 3. Jive Interface

chronized with the data structures in the source code. The user can step through the code in debug mode or workbench. This integration allows a single environment for learning data structures. The use of this tool in classroom has been an important aide for teaching students who deal with such structures.

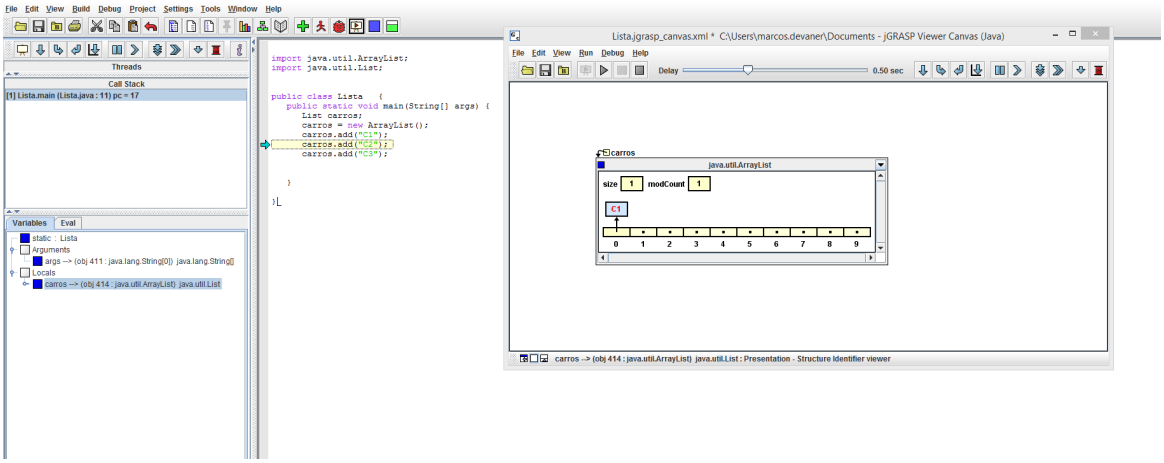

Figure 4. JGrasp Interface

Studies conducted with students indicate that the tool can have a positive and significant impact on student achievement [Cross II et al. 2007]. Pupils were more productive and more capable of detect and fix logic errors using *JGrasp* [Leal 2014].

JGrasp runs mainly in Java Swing and its components implement parts of the Java Accessibility API. This allows some elements to be available for assistive technologies. The elements include: text of the source code, text of other UI components and an alternative text to graphics components. It also produces source code and views at runtime [Cross et al. 2004]. Among other visual debuggers surveyed we can identify that this tool uses the technique of direct manipulation of objects which made it stood out from the others. In this tool using the mouse, we can drag the variable sweep into the canvas window and a Presentation viewer opens to scan as shown in Figure 4. After dragging the object into the canvas window, it's necessary to click the Step button (debug tab) until we see variables in the Variables tab.

5. The Study

Our work investigates the effects of a visual code debugger which uses the *direct manipulation technique* in the performance of DHI programmers. We compare the DHI performance in tasks with similar demands using *JGrasp* with that when they use Eclipse.

5.1. Participants and Methodology

We recruited ten deaf participants, all male, aged between 25-36 of age, all graduates from the basic Java course offered by our laboratory. In a between-subject design, the participants were ramdomly assigned to either one of two groups: the first group performed the tasks using the JGrasp tool and the second group using Eclipse.

The participants received a Java algorithm that simulates some bank transactions: *withdraw* (Subtract any value of the remaining balance) and *deposit* (Add some value to the balance exists).

Errors were deliberately included in the methods so that participants could debug the code, identify and correct them, as shown in Figures 5 and 6. Data were collected based on the analysis of the screen videos and the recorded reaction of the users during the execution of tasks.

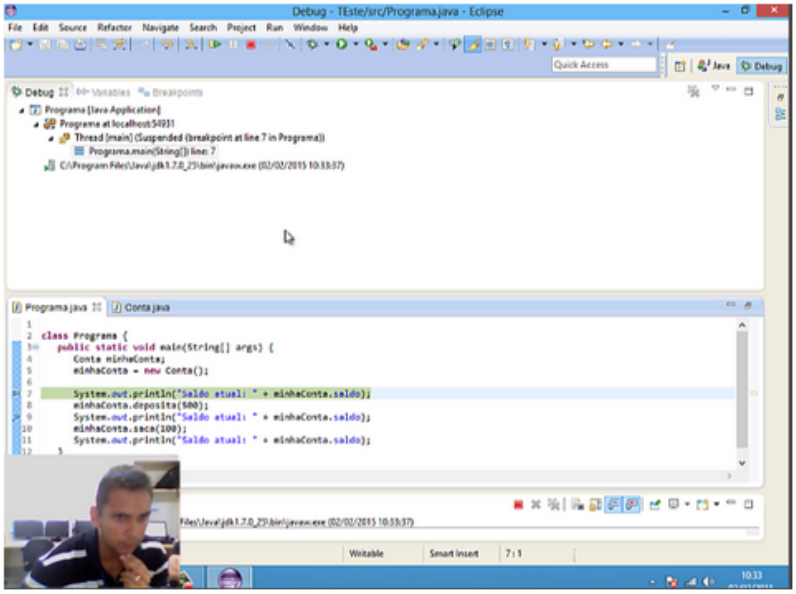

Figure 5. Subject using Eclipse IDE

A script with four tasks was given to participants to assist them in the process:

1. Adding a breakpoint in the *Withdraw()* and *Deposit()* methods.
2. Debugging to check if the return values are consistent with the objective of the method.
3. Fixing any problems found.
4. Debugging again to verify if the calculations are correct.

Methods, analysis and evaluation were applied to both experimental conditions. We use the situated testing technique to measure participants' performance when debugging task. For this analysis we observe the following variables: 1) time to complete the

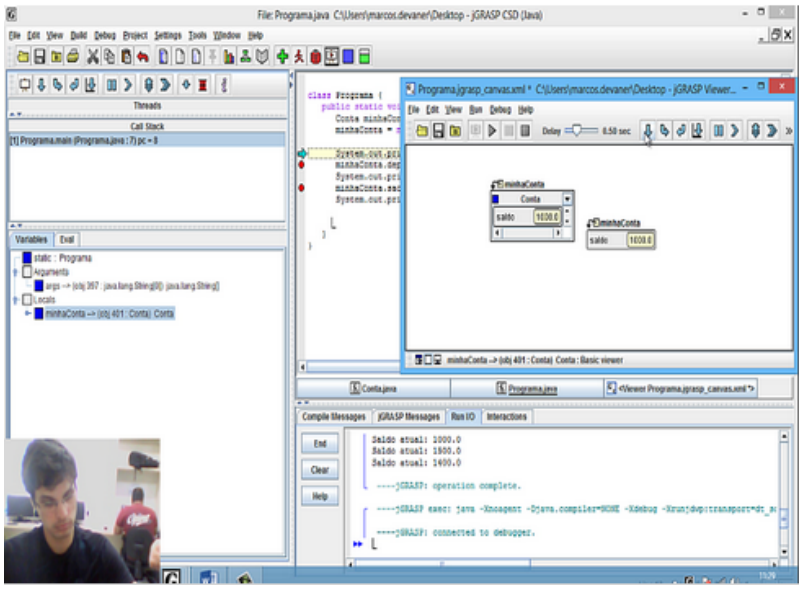

Figure 6. Subject using Jgrasp IDE

task (TCT); 2) number of times the subject asked for external help (HA) and; 3) number of tasks completed successfully (TCS).

Table 2. Results by Task - Situated Analysis

	TCS		HA		Average TCT	
Tasks	JGrasp	Eclipse	JGrasp	Eclipse	JGrasp	Eclipse
1. Adding a breakpoint in the Withdraw() and Deposit() methods.	4	5	1	2	00:26s	01:02s
2. Debugging to check if the return values are consistent with the objective of the method.	4	2	5	4	02:06s	02:51s
3. Fixing any problems found.	5	4	3	4	01:39s	02:08s
4. Debugging again to verify if the calculations are correct.	4	2	4	4	01:46s	02:45s

Besides the analysis, we applied a questionnaire based on the System Usability Scale (SUS) so that the participants could evaluate the usability of the tools. The System Usability Scale provides a reliable way to measure usability. It is a questionnaire based on the heuristics of Nielsen with five response options, where the user can strongly agree or strongly disagree withthe statements [Brooke 1996], as a Likert scale.

6. Results and Discussions

Table 2 shows the results of the variables observed during the execution of debugging tasks in JGrasp and Eclipse.

For each participant, we submitted the TCS, HA and TCT values to the one-tailed *unpaired t-test*, the results obtained are shown in Table 3.

Table 3. One-tailed *unpaired t-test* results

	t-value	p-value	significance
TCS	1.543033	0.080699	significant at $p \leq 0.10$
HA	1.206045	0.131127	not significant at $p \leq 0.10$
TCT	1.011979	0.170587	not significant at $p \leq 0.10$

We also evaluate the usability for users of the two mentioned tools using a questionnaire based on the System Usability Scale. The average SUS score for JGrasp was 72 and 50 for Eclipse. Researches indicate that a SUS score above 68 [Sauro and Lewis 2012] is considered above average.The results indicate that JGrasp tool has better usability in the users' evaluation.

We also applied the SUS scores to the two-tailed *unpaired t-test*. The *t-value* obtained was 3.0291 and the *p-value* was 0.0163. The result is therefore significant at $p \leq 0.05$. In other words, there is a significant difference in the usability of the two tools.

7. Conclusion

In this paper, we discuss various concepts and research to improve the performance of DHI programmers in tasks related to debugging. We found that systems with a visual approach combined with direct manipulation of objects produce higher productivity and usability for the DHI. Debugging is just part of the many activities a software developer is involved. The findings reported here just encourage further investigation. There is lot to be done. One thing is sure: We have to intervine in the workspace to improve productivity of the DHI programmer. How far should we use vision is a tricky question. Vision is over special resource for the DHI and we should avoid overloading it. We will carefully design a visual debugger for the DHI, having that in mind. We should also integrate this visual debugger with the *JLoad* [Silva et al. 2014]. In this way we will have an interactive development environment and Java code debugging, which followed the standards of accessibility and will allow students to create their codes in an interactive web environment, eliminating the installation and configuration of an IDE on your machine, bringing mobility for students.

References

Brooke, J. (1996). Sus - a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7.

Cattaneo, G., Faruolo, P., Petrillo, U. F., and Italiano, G. F. (2004). Jive: Java interactive software visualization environment. In *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*, pages 41–43. IEEE.

Census (2010). http://www.censo2010.ibge.gov.br/. Brazilian Institute of Geography and Statistics (IBGE).

Cross, J. H., Hendrix, D., and Umphress, D. A. (2004). Jgrasp: an integrated development environment with visualizations for teaching java in cs1, cs2, and beyond. In *Frontiers in Education, 2004. FIE 2004. 34th Annual*, pages 1466–1467. IEEE.

Cross II, J. H., Hendrix, T. D., Jain, J., and Barowski, L. A. (2007). Dynamic object viewers for data structures. *ACM SIGCSE Bulletin*, 39(1):4–8.

Cypher, A. and Halbert, D. C. (1993). *Watch what I do: programming by demonstration*. MIT press.

do Nascimento, M. D., Oliveira, F. C. d. M. B., and de Freitas, A. T. (2014). How do deaf or hearing impaired programmers perform in debugging java code? In *Anais do Simpósio Brasileiro de Informática na Educação*, volume 25, pages 593–601.

Gesueli, Z. M. and de Moura, L. (2006). Literacy and deafness: the words display - letramento e surdez: a visualização das palavras. *ETD: Educação Temática Digital*, 7(2):110–122.

Graciano, A. B. V. (2007). *Object tracking based on pattern structural recognition. - Rastreamento de objetos baseado em reconhecimento estrutural de padroes*. PhD thesis, Universidade de São Paulo.

Hutchins, E. L., Hollan, J. D., and Norman, D. A. (1985). Direct manipulation interfaces. *Human–Computer Interaction*, 1(4):311–338.

Leal, A. V. d. A. (2014). Teaching programming in high school: An approach using standards and games with concrete materials - ensino de programação no ensino médio integrado: Uma abordagem utilizando padrões e jogos com materiais concretos. Master's thesis. Available online at http://repositorio.bc.ufg.br/tede/handle/tede/3613.

Moreno, A. and Joy, M. S. (2007). Jeliot 3 in a demanding educational setting. *Electronic Notes in Theoretical Computer Science*, 178:51–59.

Rose, A., Plaisant, C., and Shneiderman, B. (1995). Using ethnographic methods in user interface re-engineering. In *Proc. DIS '95: Symposium on Designing Interactive Systems*, pages 115–122.

Santiago, V. d. A. A. (2011). The participation of deaf people in the labor market - a participação de surdos no mercado de trabalho. Available online at http://www.porsinal.pt/index.php?ps=artigos&idt=artc&cat=12&idart=299.

Sauro, J. and Lewis, J. R. (2012). *Quantifying the user experience: Practical statistics for user research*. Elsevier.

Silva, L. C., Oliveira, F. C. d., Oliveira, A. C. d., and Freitas, A. T. d. (2014). Introducing the jload: A java learning object to assist the deaf. In *Advanced Learning Technologies (ICALT), 2014 IEEE 14th International Conference on*, pages 579–583. IEEE.