Visual Debuggers and Deaf Programmers

Marcos Devaner do Nascimento[1](✉),
Francisco Carlos de Mattos Brito Oliveira[2],
Adriano Tavares de Freitas[3], and Lidiane Castro Silva[1]

[1] Computer Science Department, Ceará State University,
Itaperi Campus, Fortaleza, Brazil
marcos@projetolead.com.br, lidcastro@gmail.com
[2] Computer Science Department, University of Fortaleza, Fortaleza, Brazil
fran.oliveira@unifor.br
[3] Computing Department, Federal Institute of Ceará, Maracanaú Campus,
Maracanaú, Brazil
adriano.freitas@ifce.edu.br

Abstract. We investigated how visual debuggers impact the performance of a Java programmer who is deaf or hearing impaired (DHI). In previous work, we had shown that despite having attended accessible java course, deaf programmers still perform poorer than their hearing counterparts in tasks like debugging. In this text, we show that visual debuggers present a hope of bridging the gap between the two populations. Typical debugging tasks were assigned to both groups who used industry standard IDE (Eclipse) and a Visual Debugger (JGrasp). Qualitative and quantitative analysis show advantages for the former.

Keywords: Debugger · Accessibility · Usability · Performance

1 Introduction

To secure a position in the workplace for deaf or hearing impaired (DHI) programmers, studies must show that they have performance similar to their hearing counterparts. The *Laboratory of Distance Education for People with Disabilities* creates and offers seven courses in information technology (IT) through our accessible learning management system (LMS), among them a basic Java course using the industry-standard Eclipse programming environment. Although our LMS is equipped for the DHI and those with missing limbs, the focus of this text is on the DHI java graduate.

We are interested in empowering the DHI programmer in the daily tasks of a regular software engineer, such as software evolution, debugging. Some lessons on our Java course are reinforced by programming exercises (or programming workshops). Java workshops are implemented on LMS, which allows online collaboration between a tutor, a translator and the DHI [25]. Such collaboration is part of our strategy to create and promote a collaborative environment between a DHI programmer and a hearing coworker (tutor is not versed in sign language).

M. Antona and C. Stephanidis (Eds.): UAHCI 2016, Part III, LNCS 9739, pp. 26–37, 2016.
DOI: 10.1007/978-3-319-40238-3_3

We know that DHI graduates from our courses has inferior performance in debugging tasks when compared to their hearing counterparts who took the very same courses [11]. We expect visual debuggers and direct manipulation might improve the performance of the DHI programmer. We compare how a visual debugger (JGrasp) affects the activities of a DHI programmer in this paper. Ten participants from our basic Java course were recruited to debug code in JGrasp and Eclipse, in a between-subjects experiment. We emphasize that all subjects have already had contact with the Eclipse tool for debugging activities in our course.

Performance was measured by: (a) Time to complete the task (TCT); (b) Number of times the subject asked for external help assistance (HA) and (c) Number of tasks completed successfully (TCS). These metrics were submitted to the *Mann-Whitney U-Test*, the results were not statistically significant at $p \leq 0.05$. But, although the Eclipse tool may have be benefited due to the background of the participants, the TCT, HA and TCS variables show a similar performance between the two tools, with JGrasp showing some advantage. We can conclude that the JGrasp makes use of elements that allow a better familiarization of the features available.

A questionnaire based on the System Usability Scale (SUS) [5] was also applied. The average SUS score for JGrasp was 72 and 50 for Eclipse. The answers submitted to the *Mann-Whitney U-Test* give us a *p-value* of 0.01, a statistically significant result, thus we can conclude that JGrasp was better evaluated, usability-wise.

A simular study is presented in [11]. However, in this paper, we show some aspects of the teaching of DHI. In this way, we can justify more appropriately the use of visual tools in the deaf learning. Besides, the analysis of the results is more robust and consistent: the statistical test in the previous work was not very suitable for the experimental design and sample size; furthermore, we show, in this article, a qualitative analysis of the studied tools which allows us to understand what the DHI feel and think on the use of visual tools.

The structure of this paper is as follows: we show some aspects of deaf learning in Sect. 2. We, then present the theoretical background in Sect. 3, which deals with direct manipulation and development environments. In Sect. 4, we present related work concerning visual debuggers. Section 5 shows our study design and subject profile; and in Sect. 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 [25] and add a direct-manipulation inspired visual debugger.

2 Deaf Learning

Research shows that students with profound deafness degree have lower academic achievement compared with their hearing counterparts in all educational environments [14,18,27].

Bull and colleagues [6] show that deaf students have more difficulty in absorbing mathematical concepts when compared to listeners during their learning process.

But hearing loss can not be the cause of the problem. Nunes and Moreno [19] argue that poor mathematical performance comes from a risk factor related to time, type of education and learning opportunities for deaf students. To support this thought, Zarfaty and colleagues [29] showed that 3–4 year-old deaf children have spatial and time skills comparable to his hearing colleagues with the same age and they also have better spatial numerical skills.

Barbosa [1] argues that deaf children and listeners seem to have similar performance in cognitive functions less dependent on linguistic stimuli. For Boroditsky [4], bilingual people show divergence on the same subject when they have to approach it in both languages. This change in language also has an impact on memory. Finally, the author posits that language shapes even the most basic dimensions of human experience: space, time, causality and relationships with others.

Most of the educational materials were designed for listeners. For Perlin [21], the "listener culture" is essentially made up of auditory signals that deaf people do not use. On the other hand, the deaf use visual signals to understand the world around them. Furthermore, It is not just the lack of appropriate teaching materials, the Brazilian Sign Language (Libras) is relatively young and has a poor vocabulary, compared to Portuguese and lacks many technical terms in many areas. Online forums can be used for creating these terms. Oliveira and colleagues [20] present a comparative study on the acceptance of signs created in person and remotely. They show that acceptable and legitimate signs can also be produced using web discussions and the users can not distinguish from which method they come from.

Blatto-Vallee and colleagues [3] show that the use of visual-spatial schematic representations is a strong positive predictor of mathematical problem solving performance for deaf students. For Pinto [22], visuality seems to represent, for the deaf, the main channel for thinking and processing schemes which naturally allows the acquisition, construction and expression of knowledge, values and experiences that, otherwise, would be unreachable. The visual channel allows the reading of the deaf world and it is the support of their mental processing. The authors report the improvement self-esteem, interest and engagement among the deaf, when drawings, images and visual manipulations were used in science education, geography, art and history.

Although there are various visual Java debuggers, to our knowledge, none had been tested with the deaf. Before proceeding with the discussion of visual Java debuggers available, we must warn that users who use sign language hold less information in their short-term memory when compared to non-DHI [2]. This may be partly due to the nature of the visual-spatial information necessary for the DHI [28].

In a recent research, Sorva and colleagues [26] point out that in the last three decades, dozens of software systems have been developed in order to illustrate the runtime behavior of software for novice programmers. However, a desirable visual debugger for the deaf student should: (1) not overload the visual mode; (2) offer the programmer portability and flexibility to be used in different workplace settings; (3) have intuitive interface with accessibility features.

3 Direct Manipulation and IDE's

For Hutchins [15], direct manipulations are visual interfaces in which users operate on a representation of objects of interest. Rose [23] argues that visibility of objects and actions on these objects produce a significant difference in productivity. In some studies cited by the author, these features allow a better domain of the interface; more competence in performing tasks; ease of learning both basic and advanced functions; confidence that the user will continue to dominate the interface even if she stops using it for a while; satisfaction in using the system; increases the will of teaching others; and the desire to explore more advanced aspects of the system [23].

Some integrated development environments (IDEs) have applied not only the concept of direct manipulation as well as the concept of visualization, to making complex tasks like debugging a code simpler and more intuitive. Moreno [17] argues that 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 collegues [10] 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 abstract concepts, 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 strongly rely on visual input [12] and it is reasonable to assume that they will benefit from an IDE with these characteristics.

4 Related Work

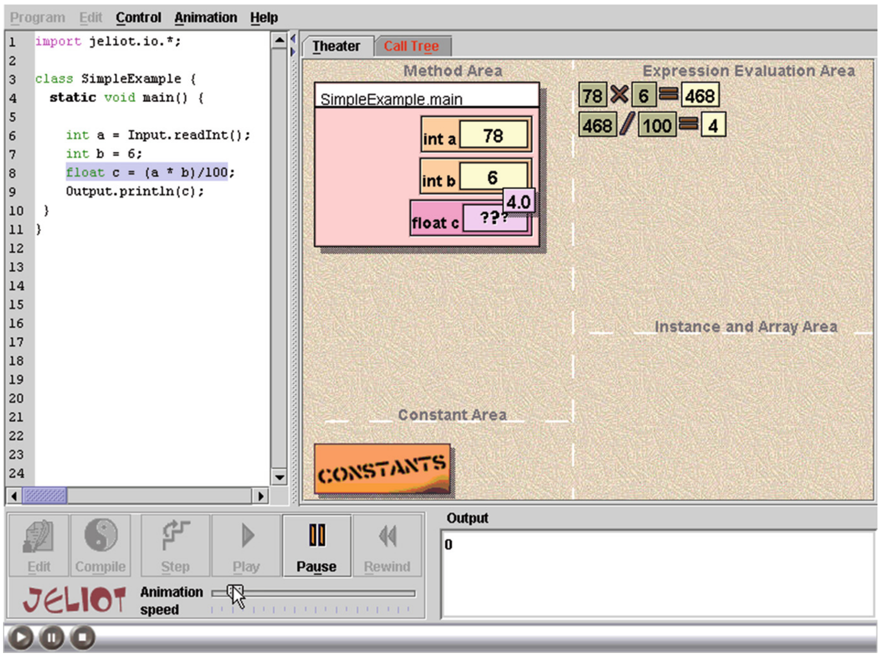
Visual debuggers implement, by its conception, direct manipulation strategies. In this section, we show some visual debuggers, briefly discuss how they implement such strategies. We also choose one of them to use in the study reported in this text and justify our choice.

4.1 Visualizing Programs with Jeliot 3

Jeliot 3 [17] 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 better understand the construction of the program.

The system is easy to use, has consistent, complete and continuous view and 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 Fig. 1.

This program seeks to be as consistent as possible in order to reduce the cognitive load of students during learning. Despite its many qualities, we evaluated that the tool has too many visual information presented at once and that



```
import jeliot.io.*;

class SimpleExample {
  static void main() {

    int a = Input.readInt();
    int b = 6;
    float c = (a * b)/100;
    Output.println(c);
  }
}
```

```
0
```

Fig. 1. Jeliot Interface

might confuse the deaf student. 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 [13].

4.2 Java Interactive Visualization Environment (Jive)

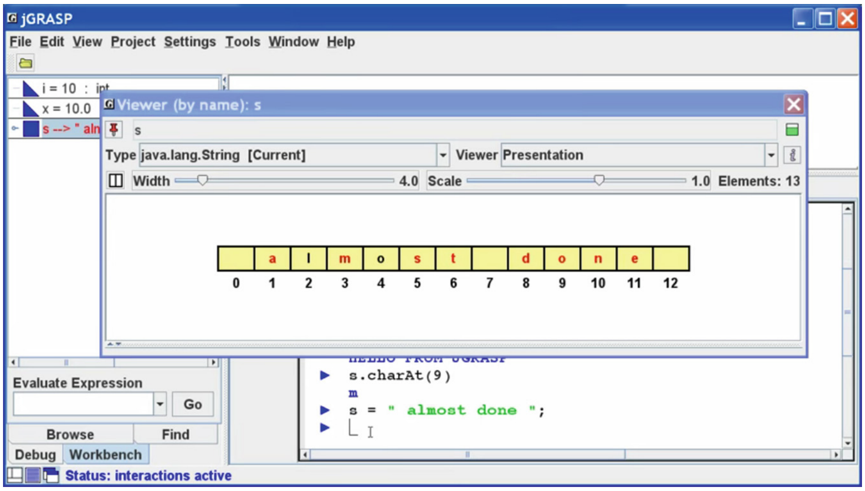
Jive [7] is an interactive execution tool developed by the Department of Science and Computer Engineering from the University of Buffalo. This system is used for: (a) debugging Java programs with rich views of object structure and interactions between methods; (b) facilitating maintenance software; providing insight into the dynamic behavior of programs and (c) teaching and learning Java.

It was originally designed as a stand-alone Java application. Later it was 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, which can generate DHI greater cognitive effort by removing the focus of logic to the concepts linked to the diagram. Therefore, we judged that it is not appropriate for the studied scenario.

4.3 JGrasp

JGrasp [8] is an IDE developed to provide dynamic and illustrative views of Java data structures. These views are generated automatically and synchronized 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, the authors claim.



```
i = 10 : int
x = 10.0
s --> " aln
```

s

Type java.lang.String [Current]   Viewer Presentation

Width 4.0   Scale 1.0   Elements: 13

	a	l	m	o	s	t		d	o	n	e	
0	1	2	3	4	5	6	7	8	9	10	11	12

```
HELLO FROM JGRASP
▶ s.charAt(9)
m
▶ s = " almost done ";
▶
```

Status: interactions active

Fig. 2. JGrasp Interface

Studies conducted with students indicate that the tool can have a positive and significant impact on student achievement [9]. Pupils were more productive and more capable of detect and fix logic errors using *JGrasp* [16].

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 [8]. 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, students can drag a variable, drop it into the canvas window and a Presentation viewer opens to scan it, as shown in Fig. 2.

5 The Study

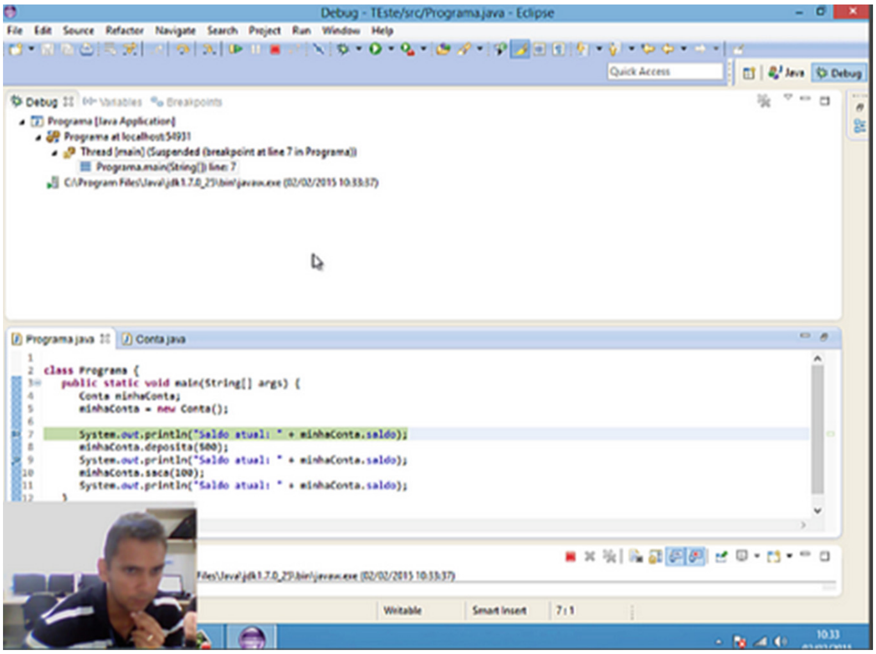
We were poised to investigate the effects of a visual code debugger, which uses the *direct manipulation technique* in the performance of DHI programmers. We compare the DHI performance in tasks using *JGrasp* and Eclipse.

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 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 Figs. 3 and 4. All trials were videotaped and partcipants interviewed for qualitative analysis. Quantitative data were extracted from video analysis.



Debug - TEste/src/Programa.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Debug
Variables
Breakpoints
Programa.java
Conta.java
Writable
Smart Insert
7 : 1
10:33

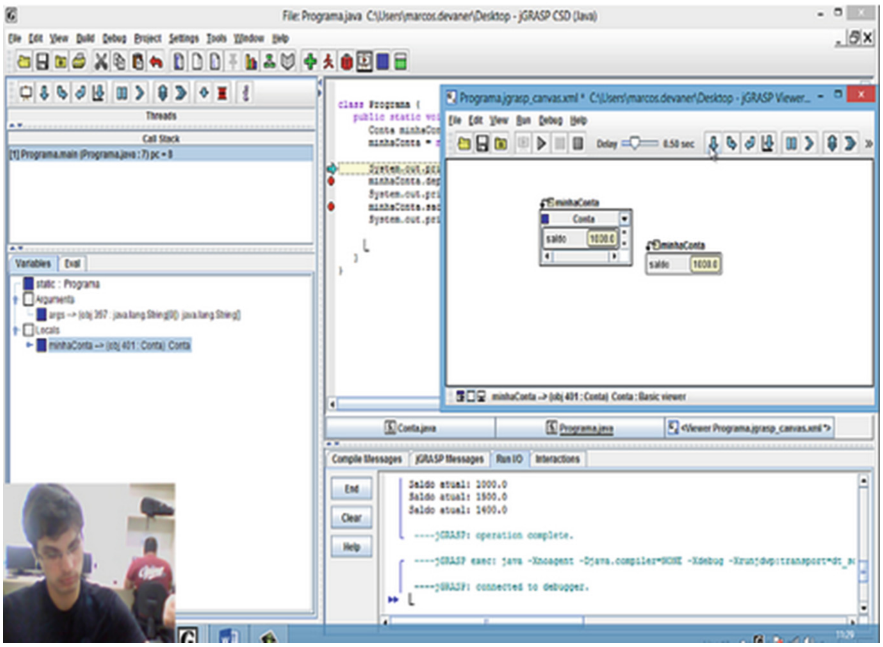
```
Programa [Java Application]
  Programa at localhost:54931
    Thread [main] (Suspended (breakpoint at line 7 in Programa))
      Programa.main(String[]) line: 7
  C:\Program Files\Java\jdk1.7.0_25\bin\javaw.exe (02/02/2015 10:33:37)
```

```java

class Programa {
    public static void main(String[] args) {
        Conta minhaConta;
        minhaConta = new Conta();

        System.out.println("Saldo atual: " + minhaConta.saldo);
        minhaConta.deposita(500);
        System.out.println("Saldo atual: " + minhaConta.saldo);
        minhaConta.saca(100);
        System.out.println("Saldo atual: " + minhaConta.saldo);
    }
```

```
Files\Java\jdk1.7.0_25\bin\javaw.exe (02/02/2015 10:33:37)
```

Fig. 3. Subject using Eclipse IDE

A script with four tasks was given to participants to assist them in the process: (a) Adding a breakpoint in the *Withdraw()* and *Deposit()* methods; (b) Debugging to check if the return values are consistent with the objective of the method; (c) Fixing any problems found; (d) Debugging again to verify if the calculations are correct.



```
class Programa {
   public static voi
      Conta minhaCon
      minhaConta = n

         System.out.pri
         minhaConta.dep
         System.out.pri
         minhaConta.sac
         System.out.pri

   }
}
```

```
Saldo atual: 1000.0
Saldo atual: 1500.0
Saldo atual: 1400.0

 ----jGRASP: operation complete.

 ----jGRASP exec: java -Xnoagent -Djava.compiler=NONE -Xdebug -Xrunjdwp:transport=dt_s

 ----jGRASP: connected to debugger.
```

File: Programa.java C:\Users\marcos.devaner\Desktop - jGRASP CSD (Java)
File Edit View Build Debug Project Settings Tools Window Help
Threads
Call Stack
[1] Programa.main (Programa.java:7) pc = 8
Variables Eval
static : Programa
Arguments
args -> {obj 357 : java.lang.String[]} java.lang.String[]
Locals
minhaConta -> {obj 401 : Conta} Conta
Programa.jgrasp_canvas.xml * C:\Users\marcos.devaner\Desktop - jGRASP Viewer
File Edit View Run Debug Help
Delay 0.50 sec
minhaConta
Conta
saldo 1600.0
minhaConta
saldo 1600.0
minhaConta -> {obj 401 : Conta} Conta : Basic viewer
Conta.java
Programa.java
<Viewer Programa.jgrasp_canvas.xml *>
Compile Messages
jGRASP Messages
Run I/O
Interactions
End
Clear
Help

Fig. 4. Subject using Jgrasp IDE

Table 1. Results by Task - Situated Analysis

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

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: (a) time to complete the task (TCT); (b) number of times the subject asked for external help (HA) and; (c) number of tasks completed successfully (TCS).

We've also applied the System Usability Scale (SUS) standard questionnaire (Likert Scale) so that the participants could evaluate the usability of the tools. SUS provides a reliable way to measure usability, is based on the heuristics of Nielsen [5].

6 Results and Discussions

6.1 Quantitative Analysis

Despite the fact that subjects were familiar with Eclipse (it was the tool used in their 120 h Java course), the numbers favor JGrasp, although not statistically significant. TCT, HA and TCS showed in Table 1 indicate that.

For each participant, we submitted the TCS, HA and TCT values to the two-tailed *Mann-Whitney U-Test*, the results obtained are shown in Table 2. All the results were not significant at $p \leq 0.05$.

Table 2. Two-tailed *Mann-Whitney U-Test* results

	U-value	Critical value	p-value
TCS	6	2 at $p \leq 0.05$	0.2113
HA	6.5	2 at $p \leq 0.05$	0.25014
TCT	7	2 at $p \leq 0.05$	0.29834

The average SUS score for JGrasp was 72 and 50 for Eclipse. Researches indicate that a SUS score above 68 [24] is considered above average. The results indicate that JGrasp tool has better usability, according to participants evaluation.

We also applied the SUS scores to the two-tailed *Mann-Whitney U-Test*. The *U-value* obtained was 1 and the *p-value* was 0.02144. 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.

6.2 Qualitative Analysis

When asked if they would like to use this system frequently, three of five participants who used JGrasp strongly agree that they would like to use the system more often. They commented that the use of the debugger increases the knowledge through practice. This is possible due to the well location of information on the screen which facilitates the use of the tool. On the other had, from the five participants who tested Eclipse, two strongly liked. From their comments, some participants had forgotten some functions and they also said that they need more practice to be able to use all features of the systems.

When questioned about complexity of the system, all participants disagreed that JGrasp is complex. Eclipse was described as complex for 40 % of its users.

In the comments, they reported lack of knowledge or difficulty in finding icons in toolbars.

We also asked if the system is easy to use. All five JGrasp users reported no difficulty. Eclipse users, on the other hand, reported that although the tabs are well positioned, the windows are not easy to interpret. They said it is due to visual appeal and one reported that the system needs to be more visually organized because this is an important issue for a deaf user.

The need of assistance to use the system was another question to the participants. Two JGrasp users agreed that they do not need help to use the system. Only one Eclipse user strongly disagrees with the need of assistance. The lack of knowledge of the system was the cause that justify the need of help to answer the tasks.

The subjects were asked whether the various system functions were well integrated. All JGrasp users reported that it is well organized and easy to use and three of them answered that the system is not complicated because it offers best practices for working with the programmer. Eclipse users commented it contains not grouped windows and features; and the icons are not interactive, therefore hindering the understanding of the tool. Three of the users said reported that the functions are far from one another.

When asked if most people would learn to use the system quickly, all users commented that it depends on development experience and training, and the learning would be fast both in JGrasp as in Eclipse. Confidence when using the system was well evaluated in both tools. They say that the more knowledgeable you are in programming, the more confident you get using the tools.

Finally we asked whether it is needed to learn many things before using the system. All users reported that they need to learn and practice before the use of the systems. Two JGrasp users strongly agree with the question, while four Eclipse users strongly agree.

The use of JGrasp platform among deaf users was more acceptable for presenting visual appeal and better distribution of functionality to the developer. Although the activities to be developed for both two groups needed the assistance for completion, we must emphasize that some users who were in the Eclipse group already knew the tool. Thus it is justified the use of JGrasp as a good debug tool for deaf.

7 Conclusion

Providing adequate learning materials, environment and tutoring is not enough to bridge performance gap between deaf or hearing impaired programmer and their hearing confederates in real world tasks. Debugging is just part of the many activities a software developer is involved in. 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 are preferred by the DHI community.

The findings reported here just encourage further investigation. There is lot to be done. One thing is sure: We have to intervene in the workspace to improve productivity of the DHI programmer. How far should we use vision is a tricky question. Vision has greater importance 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 our learning object *JLoad* [25]. 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.

We are currently investigating how several information visualization techniques can be used (alone or in a combined way) to cement that improvement and embed it in our LMS. No visual debugger was conceived with the DHI programmer in mind.

References

1. Barbosa, H.H.: Initial mathematical skills in listeners and deaf children. Cad. Cedes **91**, 333–347. Title in Portuguese: Habilidades Matemáticas Iniciais em Crianças Surdas e Ouvintes
2. Bavelier, D., Dye, M.W., Hauser, P.C.: Do deaf individuals see better? Trends Cogn. Sci. **10**(11), 512–518 (2006)
3. Blatto-Vallee, G., Kelly, R.R., Gaustad, M.G., Porter, J., Fonzi, J.: Visual-spatial representation in mathematical problem solving by deaf and hearing students. J. Deaf Stud. Deaf Educ. **12**(4), 432–448 (2007)
4. Boroditsky, L.: How language shapes thought. Sci. Am. **304**(2), 62–65 (2011)
5. Brooke, J.: Sus - a quick and dirty usability scale. Usability Eval. Ind. **189**(194), 4–7 (1996)
6. Bull, R., Blatto-Vallee, G., Fabich, M.: Subitizing, magnitude representation, and magnitude retrieval in deaf and hearing adults. J. Deaf Stud. Deaf Educ **11**(3), 289–302 (2006)
7. Cattaneo, G., Faruolo, P., Petrillo, U.F., Italiano, G.F.: Jive: java interactive software visualization environment. In: 2004 IEEE Symposium on Visual Languages and Human Centric Computing, pp. 41–43. IEEE (2004)
8. Cross, J.H., Hendrix, D., Umphress, D.A.: JGRASP: an integrated development environment with visualizations for teaching java in CS1, CS2, and beyond. In: 34th Annual Frontiers in Education, FIE 2004, pp. 1466–1467. IEEE (2004)
9. Cross, J.H., Hendrix, T.D., Jain, J., Barowski, L.: A: dynamic object viewers for data structures. ACM SIGCSE Bull. **39**(1), 4–8 (2007)
10. Cypher, A., Halbert, D.C.: Watch What I do: Programming by Demonstration. MIT press, Cambridge (1993)
11. do Nascimento, M.D., de Mattos Brito Oliveira, F.C., de Freitas, A.T: How do deaf or hearing impaired programmers perform in debugging java code? In: Anais do Simpósio Brasileiro de Informática na Educação, vol. 25, pp. 593–601 (2014)
12. Gesueli, Z.M., de Moura, L.: Literacy and deafness: the words display. ETD: Educação Temática Digital **7**(2), 110–122 (2006). Title in Portuguese: Letramento e surdez: a visualização das palavras
13. Graciano, A.B.V.: Object tracking based on pattern structural recognition: Ph.D. thesis, São Paulo University (2007). Title in Portuguese: Rastreamento de objetos baseado em reconhecimento estrutural de padrões

14. Gregory, S.: Mathematics and deaf children. Issues Deaf Educ. 119–126 (1998)
15. Hutchins, E.L., Hollan, J.D., Norman, D.A.: Direct manipulation interfaces. Hum. Comput. Interact. **1**(4), 311–338 (1985)
16. de Aquino Leal, A.V.: Teaching programming in high school: An approach using standards and games with concrete materials. Master's thesis (2014). Title in Portuguese: Ensino de Programação no Ensino Médio Integrado: Uma Abordagem Utilizando Padrões e Jogos com Materiais Concretos. http://repositorio.bc.ufg.br/tede/handle/tede/3613
17. Moreno, A., Joy, M.S.: Jeliot 3 in a demanding educational setting. Electron. Notes Theor. Comput. Sci. **178**, 51–59 (2007)
18. Nogueira, C.M.I., Zanquetta, M.E.M.: Deafness, bilingualism and traditional teaching of mathematics. Zetetiké: Revista de Educação Matemática **16**(30), 219–237 (2009). Title in Portuguese: Surdez, bilingüismo e o ensino tradicional de Matemática: uma avaliação piagetiana
19. Nunes, T., Moreno, C.: Is hearing impairment a cause of difficulties in learning mathematics. Dev. Math. Skills, 227–254 (1998)
20. de Oliveira, F.C., Gomes, G.N., de Freitas, A.T., de Oliveira, A.C., Silva, L.C., Queiroz, B: A comparative study of the acceptability of signs for the brazilian sign language created in person and remotely. In: Proceedings of the 46th ACM Technical Symposium on Computer Science Education, pp. 207–211. ACM (2015)
21. Perlin, G.: The place of deaf culture. A invenção da surdez: cultura, alteridade, identidade e diferença no campo da educação, pp. 73–82 (2004). Title in Portuguese: O lugar da cultura surda
22. de Souza Pinto, M.A., dos Santos Gomes, A.M., Nicot, Y.E.: The visual experience as a facilitator in science education for deaf students. Revista Areté: Revista Amazônica de Ensino de Ciências **5**(09) (2014). Title in Portuguese: A experiência visual como elemento facilitador na educação em ciências para alunos surdos
23. Rose, A., Plaisant, C., Shneiderman, B.: Using ethnographic methods in user interface re-engineering. In: Proceedings of the DIS 1995: Symposium on Designing Interactive Systems, pp. 115–122 (1995)
24. Sauro, J., Lewis, J.R.: Quantifying the User Experience: Practical Statistics for User Research. Elsevier (2012)
25. Silva, L.C., de Oliveira, F.C., de Oliveira, A.C., de Freitas, A.T.: Introducing the jLoad: a java learning object to assist the deaf. In: 2014 IEEE 14th International Conference on Advanced Learning Technologies (ICALT), pp. 579–583. IEEE (2014)
26. Sorva, J., Karavirta, V., Malmi, L.: A review of generic program visualization systems for introductory programming education. ACM Trans. Comput. Educ. (TOCE) **13**(4), 15 (2013)
27. Traxler, C.B.: The stanford achievement test: national norming and performance standards for deaf and hard-of-hearing students. J. Deaf Stud. Deaf Educ. **5**(4), 337–348 (2000)
28. Wilson, M., Emmorey, K.: Comparing sign language and speech reveals a universal limit on short-term memory capacity. Psychol. Sci. **17**(8), 682–683 (2006)
29. Zarfaty, Y., Nunes, T., Bryant, P.: The performance of young deaf children in spatial and temporal number tasks. J. Deaf Stud. Deaf Educ. **9**(3), 315–326 (2004)