A Comparative Study of Deaf and Non-deaf Students' Performance When Using a Visual Java Debugger

Marcos Devaner do Nascimento*, Francisco Carlos de M. B. Oliveira†, Shara Shami Araújo Alves§, Adriano Tavares de Freitas§,¶, Luiz Alexandre C. B. Gomes* and Anderson Severo de Matos¶
*Ceará State University, Itaperi Campus
Computer Science Department, Fortaleza, Brazil
Email: marcos@projetolead.com.br
†University of Fortaleza
Computer Science Department, Fortaleza, Brazil
Email: fran.oliveira@unifor.br
‡Federal Institute of Ceará, Fortaleza Campus
Teleinformatics Department, Fortaleza, Brazil
§Federal Institute of Ceará, Maracanaú Campus
Computing Department, Maracanaú, Brazil
¶Federal University of Ceará, Computing Department, Fortaleza, Brazil

***Abstract*—IT job market seems to be an interesting alternative for those who are deaf or hearing impaired (DHI). All accessible (interface+content+learning objects) online courses have the reach and the means to, potentially, impact the status quo. All lab offers such courses to both DHI and non-DHI communities. Still, DHI graduates legs behind non-DHI in debugging-related tasks when the industry-standard Eclipse (Development Environment) is used [1]. Visual debuggers improve DHI performance [2]. In this paper, we present Java Accessible Visual Debugger (JAD). JAD was conceived to be the first accessible visual debugger and to leverage debugging performance between DHI and non-DHI. This paper presents a study involving DHI/non-DHI and JAD. Three metrics were used to assess performance: 1) number of fully completed tasks (TFC); 2) time to task completion (TCT); 3) number of external help requests (HR). Non-paired t-Student's tests show no significant difference in any of the above criteria.**

I. Introduction

According to the demographic census from 2010 [3], Brazil has 9.7 million citizens who are deaf or have some kind of hearing impairment, which represents 5.1% of the whole country population. 100+ employee companies are required by federal law to have people with disability (PWD) in their workforce. Such law represents an opportunity for improving the quality of life for the PWD. Jobs at IT industry can require less training and, normally, there are vacancies.

Our lab provides free online java courses, mostly IT related (java programming, database administration, etc.). Content, technology and tutoring are made accessible for the PWD. Both PWD and non-PWD can enroll on our courses, which are primarily aimed at people in social-economic vulnerability.

The studies presented and discussed on this paper involve deaf or hearing impaired (DHI) java programming graduates. We assess how they perform in real world tasks. We try to show to prospective employers that there's no need to discriminate DHI programmers, as they perform equally, mathematically speaking, when compared to the non-DHI (or non-PWD). Unfortunately, this is not the case in tasks involving software evolution (debugging) and DHI/Non-DHI java programming graduates, despite all the effort to provide proper accommodation [4], more specifically in debugging activities.

We have observed that DHI pupils struggle to understand and manipulate development tools, such as IDEs (Integrated Development Environments) [4]. Our online learning platform is accessible, the IDE is not. We use the java industry standard, Eclipse [1]. In a follow-up study [2], we did a comparative analysis of a DHI students group over the use of Eclipse IDE and a similar tool called JGrasp [5], that offers a visual debug feature. We wondered how visual debuggers would impact DHI performance as proper visualizations might improve the situation (Section 2). Performance in both studies was measured in: 1) number of tasks completed successfully (TCS); 2) time to task completion (TCT); 3) number of external help assistences (HA). Although not statistically significant, the numbers favor the visual debugger. We also performed a usability assessment, applying the System Usability questionnaire Scale - SUS [6] for both IDEs. The average SUS score for JGrasp was 72 and 50 for Eclipse. We also applied the SUS scores to the two-tailed *Mann-Whitney U-Test* and the obtained p-value was 0.02144. The result is therefore significant at $p \leq 0.05$.

Despite the promising results, neither JGrasp nor any other visual debugger we found were designed with the DHI in mind. In this paper, we introduce our visual accessible Java debugger JAD. JAD allies the Web Content Accessibility

978-1-5090-5920-1/17/$31.00 ©2017 IEEE

Guidelines (WCAG) [7] with information visualization (INFOVIS) techniques. In this paper we compare DHI and non-DHI JAD debugging performance using the same metrics used in previous studies. The results were also promising, as this time, no statistically significant difference across how the DHI/non-DHI groups perform was found.

This paper is divided as follows: Section II shows the theoretical foundations for this work; Section III discusses some related projects and studies; JAD visual debugger itself is presented in Section IV. In Section V we present the studies (design, methodology, and data collection, result presentation) and discuss the obtained results and then, we draw conclusions in Section VI.

II. Theoretical Foundation

Over this section we discuss issues associated with the learning process for the DHI: How they learn and how the design of appropriate material impacts learning. We then present and debate some information visualization techniques as we understand they might be more engaging for that population.

A. The DHI Learning Process

Studies suggest that, when compared to non-DHI people, DHI students have lower academic performance on every education environment [8], [9], [10]. The DHI students manifest difficulty on internalizing mathematical concepts.

However, the lack of hearing capabilities itself might not be the source of the problem, because the low mathematical accomplishment is also related to time, type of education and learning opportunities given to the deaf student [11]. Zarfaty et al. showed that 3 and 4 years old DHI children have spatial and temporal/chronological skills comparable to their non-DHI friends at same age, and even better spatial numerical skills [12]. Barbosa argues that non-DHI and DHI children have similar performance on less linguistic dependent cognitive functions [13]. In [14], Boroditsky draws attention to the fact that bilingual people diverge over the same subject when trying to explain it in both languages. This language switching process impacts memory. Finally, the author says that the language shapes even the more fundamental dimensions of human experience such as space, time, causality and relationships. We argue that having an environment with more visual cues and with the addition of sign language support, DHI might have a better chance of understanding.

Software and educational material are normally produced with a non-DHI public on mind. Therefore, people with hearing impairment need to put an extra effort to understand a specific subject or to adapt to a software. Perlin believes that the listener culture is essentially composed by audio signals that deaf people do not use, because they rely on visual signals to comprehend the world around them [15]. Besides the lack of adequate educational content, Brazilian Sign Language (Libras) has been officialized only recently and by so is still needing some expressions for specific fields. All these issues contribute to add extra difficulties for the DHI learning and performing, in general.

B. Information Visualization as an Strategy to Improve Deaf Fulfillment

Blatto-Vallee and colleagues show that the use of schematic visual and spatial representations is a strong positive performance predictor on math problems resolution for deaf students [16]. In [17], the visualization seems to represent the main thinking and processing channel for deaf people, allowing knowledge acquisition, construction and expression. The authors report improvements on self esteem, interest and engagement with other deaf people when draws and manipulable images were used in the sciences, geography, arts and history courses.

Studies point out the DHI usually are less capable of retaining information when compared to non-DHI [18]. This might be partially due to the need of visual and spatial nature on information from the DHI [19].

Software visualizations to help beginners understand program behavior are not new [20]. A desirable visual debugger should be cautiously crafted as it must: 1) avoid abusing the use of visual mode, as that might lead to undesired sensorial overload and more confusion; (2) provide an intuitive UI (user interface) with accessible features; (3) give programmers enough portability and flexibility to work with this tool in different configurations and workplaces. We understand that an IDE with such qualities may reduce performance difference when compared to non-DHI.

III. Related Work

A. Codemap

Codemap is an Eclipse plugin that provides spatial visualization for code based on cartographic schemes [21]. Code is presented in blocks, which are separated from each other considering the lexical similarity of both features of code and structural dependencies. Bigger classes (larger number of lines of code) cover bigger parts of the scenario. Distance between clusters are calculated based on their own structural dependencies, as shown in Figure 1.

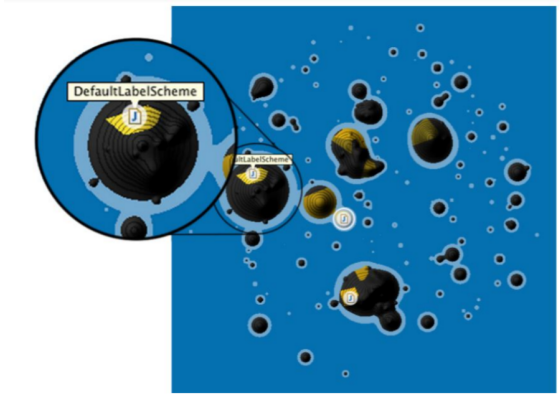
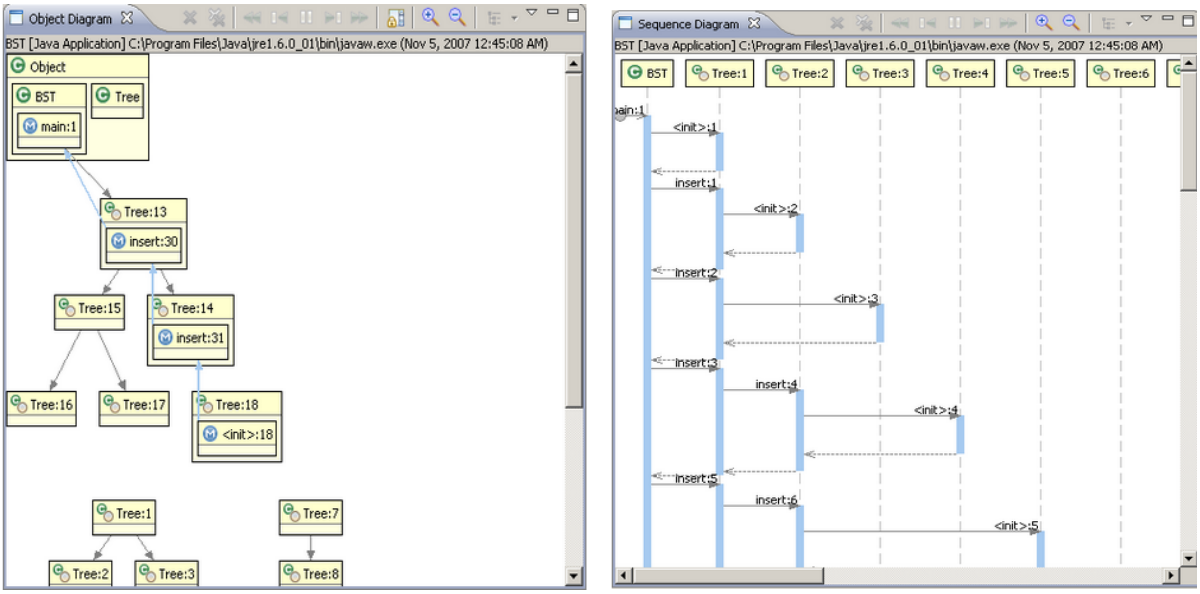


Fig. 1: Codemap visualization for a set of source code files.



(a) Object Diagram

(b) Sequence Diagram

Fig. 2: JIVE

It was observed that programmers often became lost on their own code lines, and that the use of textual references consumed extra cognitive resources. [22]. It seems that the proposal and employment of visualizations for improving the understanding software code is tricky and might add undesired effort.

B. JAVA Interactive Visualization Environment - JIVE

Developed at Buffalo University, JIVE is an interactive execution tool, which employs visualization techniques for object structures and interactions between methods, facilitating code evolution, and providing information about program's behavior during execution [23].

It was originally designed to be an autonomous Java application. Recently, it went through a major review and now is comprised of a set of plug-ins and is distributed with Eclipse. It offers two perspectives to visualize code execution: Object diagram and sequence diagram as shown in Figure 2.

To fully benefit from such visualizations, students should be familiar to UML diagrams. Although we seek to understand how visualizations impact DHI java programmers, UML diagrams are not necessarily easy to understand, as they normally have too much information to display. That might be avoided as we do not want to overload much precious visual mode for the DHI with extra information.

C. Java Learning Object to Assist the Deaf - JLoad

Our lab's JLoad (Figure 3) is a learning object conceived and built under the universal design paradigm aimed to help teaching Java first lessons to DHI students. JLoad is embedded on our online accessible learning platform. Java course writers create workshops, a series of step-by-step instructions to program creation conveyed both in Portuguese and Libras. Using JLoad, students can submit their work either partially or as a whole to assessment (grading or feedback-only proposes). Tutors have access to pupils' workspaces facilitating communication between a DHI student and a non DHI instructor. DHI students can always request online help from a signer to mediate the asynchronous conversation between them and tutors. In this two or three party chat, students can highlight the portion of the source code he or she has questions about [24].

JAD is integrated to JLoad. By clicking the "debug" button, the JAD debugging perspective is opened. This integration allows students to still have access to the JLoad interactive features while debugging some code. Both JLoad and JAD combined constitute an integrated environment that is intuitive, accessible and interactive.

D. JGrasp

JGrasp is visual debugger also aimed to helping students better understand Java intricacies [5]. It provides dynamic and illustrative structure visualization for data and Java objects. Such visualizations are automatically generated and displayed in a panel during execution. These visual representations are synchronized with source code structure. Studies suggests that students were more productive and capable of detecting and correcting errors using JGrasp, when compared to no use of a support tool [5]. JGrasp relies on Java Swing for UI, implementing parts of the Java Accessibility API (JAAPI).

JGrasp's visualizations are accompanied with direct manipulation [25], as objects, visually represented, can be moved around in the environment. It is possible to use the mouse to drag an object into a window when more detailed exploration becomes available, as shown in Figure 4.

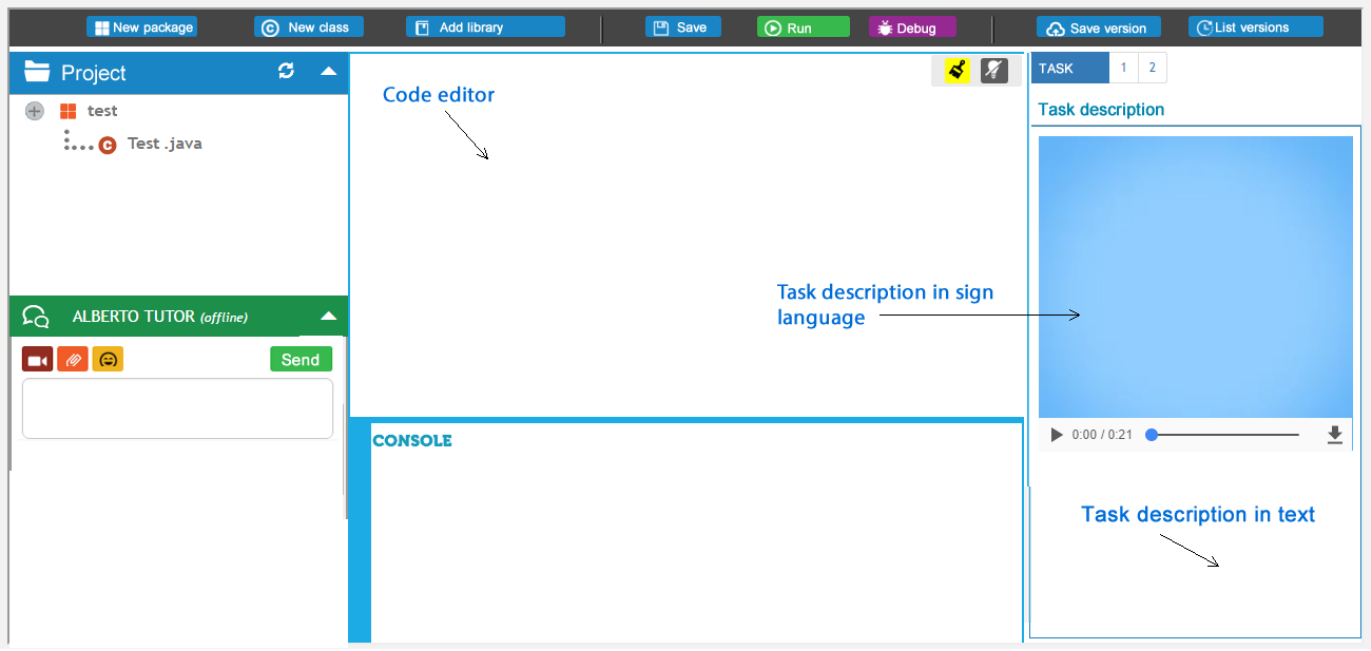


Fig. 3: JLoad UI.

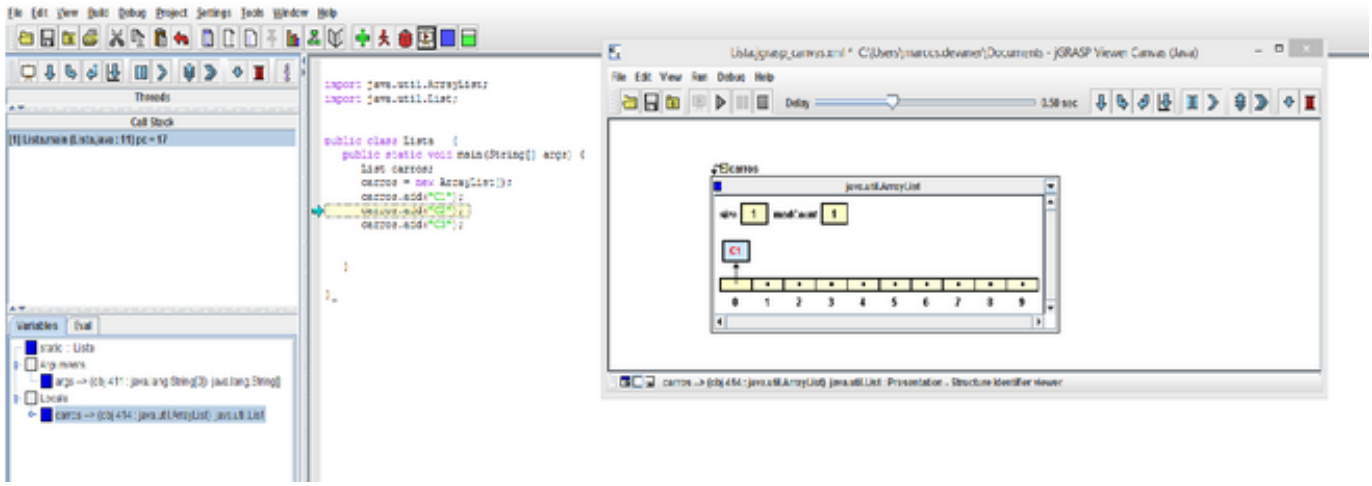


Fig. 4: JGrasp UI.

IV. Visual Accessible Debugger Proposal

Since JAD is embedded in JLoad, which in turn is a learning object. Such embedding is convenient for the pupils because they can easily start debugging from JAD, while doing their workshops. JAD user interface shown in Figure 5 was projected following the results of previously mentioned user studies. One can see accessible chat window on the lower left hand side of JLoad/JAD environment. Pupils can always ask for help in sign language and use the code highlighting feature to make direct references in their discourse. Java codes are versioned and stored in the platform and made available for both students and tutors. It provides control buttons such as debug, pause, next line and previous line. Among the implemented direct manipulation techniques, JAD offers floating panels for variables inspection that can be moved throughout the screen, and also the option to hide/show this information [25].

JAD allows students to perform the debugging process line by line without the need to define breakpoints, as in traditional debuggers. This should facilitate the use of the debugger by beginners. JAD allows debuggers to return to a previous visited line during execution, automatically reverting the variables to their early values, giving students more opportunities to understand program behavior Figure 5.

A. JAD UI and Variable Visualization

Variables are presented in two ways on JAD: 1) Default view, as shown in Figure 6 on the right, where a list of

Fig. 5: JAD UI.

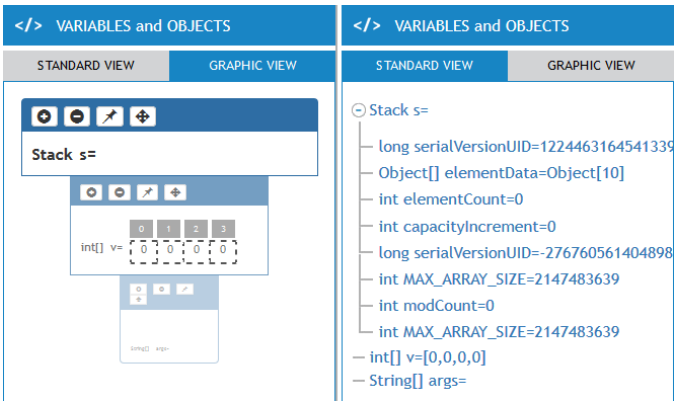


Fig. 6: Variables visualization panels.

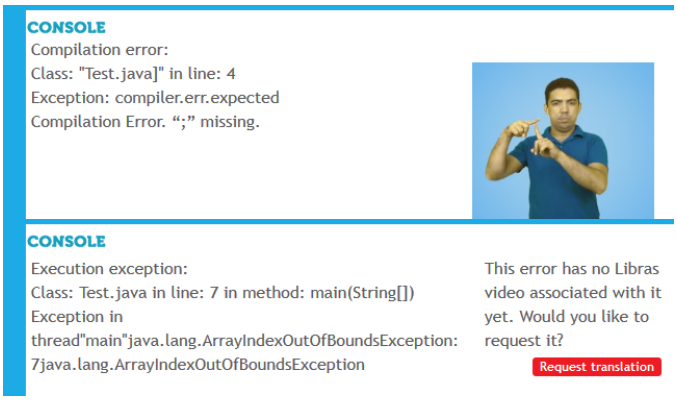


Fig. 7: Error floating panel.

variables is displayed and updated according to their life cycle and 2) the Canvas based view, as shown in Figure 6 on the left, where the variables are presented using the fisheye technique [26] along with mapping and visual transformations strategies. These techniques allow a more detailed vision of an area of the source code without losing visual cues that help keeping the programmer's bearings.

The most recent variable used on the debugging process appears on top, while the first ones are moved to behind the variable's panels stack, losing focus and opacity to help users focusing on the current variable. Data structures such as stack, queue or list are presented on this variable panel, each of which with their own visual representation.

B. Code Execution and Errors Handling

Errors/messages are presented both in Brazilian Portuguese and Libras. These error messages are not only mere translations from the original Java machine, which are normally hard to understand. JAD messages can be reconfigured to any other sign or written language as wished by course designers and tutors. Our error messages translations were written along with examples that can help students to identify what might had caused the problem as one can see in Table I.

The error floating panel (Figure 7) shows: 1) the thrown exception name; 2) the class where it occurred; 3) the line where it was thrown and; 4) possible errors on the code that might had led to the exception.

We have invested to make JAD as accessible as possible. Several WCAG recommendations were followed (Table II), not only for the DHI, but also for the blind or visually impaired, those with missing limbs, etc. Some functional features are shown in Table III.

TABLE I: Examples of exceptions adapted to a non-sign and sign language

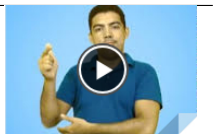
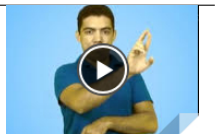
Exception in Java	**Error in English**	**Non-sign language contextual info**	**Sign Language contextual info**
compiler.err.expected	';' expected	Compilation Error. ; missing.	
java.lang.ArrayIndex OutOfBoundsException	Exception in thread"main"java.lang. ArrayIndexOutOf BoundsException: 4	Execution Error. Non-existent vector index. Check [].	
Error: compiler.err.prob. found.req	possible loss of precision found: double required: int	Compilation Error. You are trying to copy an integer to double. Check variable definitions.	

TABLE II: Accessibility features - Comparative

Accessibility feature	**JAD**	**JGrasp**	**JIVE**	**Codemap**
Shortcuts	x	x		
Auto contrast adjustment	x			
Font resizing	x	x		
Pre-recorded videos in Libras	x			
Screen reader support	x	x		

TABLE III: Functional features - Comparative

Functional features	**JAD**	**JGrasp**	**JIVE**	**Codemap**
Portability	x			
Reverse Stepping	x		x	
INFOVIS Techniques to construct objects and the relation between them	x	x	x	x
CHAT with the option of recording videos for sign language	x			
Translation requests for errors or exceptions	x			
Administrator profile to manage translations	x			
Developed code versions	x			
Code marking for further review	x			

V. User Studies

A. Subject recruitment, study design and methodology

We invited our java basic course graduates (both DHI and non-DHI) to participate on this study. Five DHI and five non-DHI answered our call (all male, from 26 to 51 years of age). Participants from both groups were asked to fix errors in two java classes (one logic, one syntax) using JAD. Participants should debug, find, fix the errors, run the fixed program and report its output. The code itself (shown in Figure 8) was very simple, a calculator.

For tool evaluation, three metrics were used: time to complete the task (TCT); if the all tasks were fully completed (TFC) and the number of external help requests during the experimentation (HR). The obtained data for each metric was submitted to the *Shapiro-Wilk* normality test [27]. Then the *non paired t-Student test* was applied with 95% reliability range to compare each group answers.

Subjects were also required to answer a post-study questionnaire (SUS, [6]) and encouraged to comment on their answers.

B. Results

1) Quantitative Analysis: The results are presented in the Table IV.

TABLE IV: Result of the metrics

metric	**p-value**	**error**
TCT	0.56	5%
TFC	0.55	5%
HR	0.46	5%

The p-values show no major differences between DHI and non-DHI students on the applied metrics TCT, TFC and HR. In previous work, when we compare DHI and non-DHI analysing the same metrics over the use of Eclipse debugging perspective, the significant difference becomes noticeable. It was found significant statistical difference in the TCT ($p = 0.0153$) and TFC ($p < 0.001$) metrics [4].

2) Qualitative Analysis: From the survey application results, the users could report some difficulties and improvements. The tool itself was well accepted by both DHI and non-DHI. DHI users felt better suited by the tool in comparison to other tools, and reported that JAD is easy to use, simple

```java
public class TesteCalculadora{
   public static void main(String[] args) {
      Calculadora calc = new Calculadora();

      calc.somar(3,2);
      System.out.println(calc.total);

      calc.dividir(10,2);
      System.out.println(calc.total);

   }
}
```

```java
public class Calculadora{
   int total;

   int somar(int valor1, int valor2){
      this.total = valor1 - valor2;
      return total/0;
   }

   int dividir(int valor1, int valor2){
      this.total = valor1 / valor2;
      return total/0;
   }
}
```

Fig. 8: Java source code.

and intuitive, and it helped a lot on the debugging process. Usability issues were noted and are listed below:

- Some users reported the lack of button descriptions, although the presence of a hint (upon mouse focus) was pointed as a good choice for description completion;
- Buttons on toolbar were proposed to be joined into a button group;
- Depending on screen resolution it was not possible to see the console tab. So we will try to include this component in a more responsible fashion way to assure it become always accessible;
- Deaf users reported the Libras video was a bit small and lacks the full-screen option;
- Some users proposed modifications for buttons and other components position on screen, along with the presence of explicit description for each one;
- Many participants committed the same mistake of trying to debug the code before saving it;
- Users suggested a short video lesson to better explain a thrown exception, providing few examples and code correction proposals.

VI. Discussion and Conclusion

In previous research, we showed that DHI java graduates perform more poorly in software evolution tasks, particularly debugging, when compared to their hearing comrades [4]. We have also shown that visual debuggers impact positively DHI debugging performance, in accordance with the metrics we proposed on our first study [2].

Based on previous experiences, both ours and from other colleagues, JAD is conceived and built as a web debugger, visually oriented and accessible, making use of information visualization techniques.

Armed with the same metrics used in all previous studies, and with the help of 5 DHI and 5 non-DHI, we investigated how an accessible visual debugger impacts their behavior in tasks related to software maintenance and also improving the tool.

A non-paired t-Student's (alpha set at 0.05) tests show no significant difference in any of the metrics: 1) Number of fully completed tasks (TFC) (p-value, 0.55); 2) time to complete the tasks (TCT) (p-value, 0.56); and 3) number of help requests (HR) (p-value, 0.46). From the results, it is possible to say that there are no significant differences between DHI and non-DHI people use of the tool. This scenario suggests we managed to enhance deaf students performance when compared to non-deaf ones, at same time the last group also benefits from the inclusive features aimed to deaf users.

It seems that JAD has potential on learning support, for both deaf and non-deaf students, specifically on the programming languages learning process. Both JAD and JLoad tools might become an inclusive IDE that could be used even outside the learning context, in actual workplaces. For further work, we are planning to better explore the web characteristics of this tool to evolve it in term of collaboration between deaf and non-deaf people, getting even closer to real world scenarios.

References

[1] I. F. Eclipse, "Eclipse - the eclipse foundation open source community website." 2016. [Online]. Available: https://eclipse.org/

[2] M. D. do Nascimento, F. C. d. M. B. Oliveira, A. T. de Freitas, and L. C. Silva, "Visual debuggers and deaf programmers," in *International Conference on Universal Access in Human-Computer Interaction*. Springer, 2016, pp. 26–37.

[3] Census, http://www.censo2010.ibge.gov.br/, 2010, brazilian Institute of Geography and Statistics (IBGE). [Online]. Available: http://www.censo2010.ibge.gov.br/

[4] M. D. do Nascimento, F. C. d. M. B. Oliveira, and A. T. de Freitas, "How do deaf or hearing impaired programmers perform in debugging java code?" in *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, vol. 25, no. 1, 2014, p. 593.

[5] A. V. d. A. Leal, "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. Available at http://repositorio.bc.ufg.br/tede/handle/tede/3613.

[6] J. Brooke *et al.*, "Sus-a quick and dirty usability scale," *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.

[7] W. W. W. Consortium *et al.*, "Web content accessibility guidelines (wcag) 2.0," 2008.

[8] S. Gregory, "Mathematics and deaf children," *Issues in deaf education*, pp. 119–126, 1998.

[9] C. B. Traxler, "The stanford achievement test: National norming and performance standards for deaf and hard-of-hearing students," *Journal of deaf studies and deaf education*, vol. 5, no. 4, pp. 337–348, 2000.

[10] C. M. I. Nogueira and M. E. M. Zanquetta, "Deafness, bilingualism and traditional teaching of mathematics." *Zetetiké: Revista de Educação Matemática*, vol. 16, no. 30, pp. 219–237, 2009, title in Portuguese: Surdez, bilingüismo e o ensino tradicional de Matemática: uma avaliação piagetiana.

[11] T. Nunes and C. Moreno, "Is hearing impairment a cause of difficulties in learning mathematics," *The development of mathematical skills*, pp. 227–254, 1998.

[12] Y. Zarfaty, T. Nunes, and P. Bryant, "The performance of young deaf children in spatial and temporal number tasks," *Journal of Deaf Studies and Deaf Education*, vol. 9, no. 3, pp. 315–326, 2004.

[13] H. H. Barbosa, "Initial mathematical skills in listeners and deaf children." *Cad. Cedes*, vol. 33, no. 91, pp. 333–347, 2013, title in Portuguese: Habilidades Matemáticas Iniciais em Crianças Surdas e Ouvintes.

[14] L. Boroditsky, "How language shapes thought," *Scientific American*, vol. 304, no. 2, pp. 62–65, 2011.

[15] G. Perlin, "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.

[16] G. Blatto-Vallee, R. R. Kelly, M. G. Gaustad, J. Porter, and J. Fonzi, "Visual–spatial representation in mathematical problem solving by deaf and hearing students," *Journal of Deaf Studies and Deaf Education*, vol. 12, no. 4, pp. 432–448, 2007.

[17] M. A. d. S. Pinto, A. M. d. S. Gomes, and Y. E. Nicot, "The visual experience as a facilitator in science education for deaf students," *Revista Areté: Revista Amazônica de Ensino de Ciências*, vol. 5, no. 09, 2014, title in Portuguese: A experiência visual como elemento facilitador na educação em ciências para alunos surdos.

[18] D. Bavelier, E. L. Newport, M. L. Hall, T. Supalla, and M. Boutla, "Persistent difference in short-term memory span between sign and speech: Implications for cross-linguistic comparisons," *Psychological Science*, vol. 17, no. 12, pp. 1090–1092, 2006.

[19] M. Wilson and K. Emmorey, "Comparing sign language and speech reveals a universal limit on short-term memory capacity," *Psychological Science*, vol. 17, no. 8, pp. 682–683, 2006.

[20] J. Sorva, V. Karavirta, and L. Malmi, "A review of generic program visualization systems for introductory programming education," *ACM Transactions on Computing Education (TOCE)*, vol. 13, no. 4, p. 15, 2013.

[21] A. Kuhn, D. Erni, and O. Nierstrasz, "Towards improving the mental model of software developers through cartographic visualization," *arXiv preprint arXiv:1001.2386*, 2010.

[22] R. DeLine, A. Khella, M. Czerwinski, and G. Robertson, "Towards understanding programs through wear-based filtering," in *Proceedings of the 2005 ACM symposium on Software visualization*. ACM, 2005, pp. 183–192.

[23] G. Cattaneo, P. Faruolo, U. F. Petrillo, and G. F. Italiano, "Jive: Java interactive software visualization environment," in *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*. IEEE, 2004, pp. 41–43.

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

[25] S. K. Card, J. D. Mackinlay, and B. Shneiderman, *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

[26] G. W. Furnas, *Generalized fisheye views*. ACM, 1986, vol. 17, no. 4.

[27] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965.