2014 International Conference on Advances in ICT for Emerging Regions (ICTer): 035 - 042

Use of Audio in Software Visualization

Marie Olsson[#1]

[#] *Department of Computer and Systems Sciences*
Stockholm University, Box 7003, 164 07 Kista, Sweden
[1]marieols@dsv.su.se

***Abstract*— The concepts and processes of computer programs and algorithms are often found difficult to learn and teach. However, software visualization is an alternative tool to facilitate learning and teaching of those abstract and dynamic entities. In this study, experiments and interviews were carried out to explore the perceived utility of combining animations and text with audio in software visualization. The study thus examines whether learning and teaching can be improved regarding the understanding of basic concepts of object-oriented programming for beginner students by adding recorded verbal explanations. Two kinds of software visualization – program visualization and algorithm visualization – were investigated.**

***Keywords*— Software visualization, student learning, student perspective.**

I. INTRODUCTION AND AIM

As reported in literature on educational programming [1], computer programs and algorithms are both abstract and dynamic entities that are often difficult to learn and teach. It is possible to facilitate teaching and learning in multiple ways in computer programming, software visualization being one of them.

Software visualization can be seen as the use of graphical representations, in the form of text, images and animations. Two basic concepts of software visualization are program visualization and algorithm visualization. *Program visualization* is a graphical representation of a running program, i.e. program code execution. *Algorithm visualization* is a graphical representation of an algorithm, i.e. a set of instructions that will perform a specific task.

If visualization is combined with verbal information (audio), the transfer of knowledge will become more effective according to dual-coding theory. Dual-coding is not a theory specifically used in computer science and programming education. However, it is interesting to apply the theory in programming education.

Software visualization can be seen as the utilization of graphical representations in form of text, pictures and animations. An animation is a rapid display of a sequence of pictures, also often combined with text.

In this study audio is added to the software visualizations. As far as is known, text, animation and audio has not all been tried together in controlled experiments regarding object-oriented programming.

The combination of animation and audio is expected to improve the teaching and learning processes in this study. It is also interesting to investigate whether audio combined with animation and text, will further enhance the learning and teaching processes.

There is presently a lack of knowledge about how useful a combination of animation, text and sound is in the teaching and learning of basic concepts in object-oriented programming for beginner students. The relationship between the resource in form of animation, text and sound, and the resulting outcome from the utilization of that resource is examined in this study. Accordingly, a student's improved understanding of the outcome of that specific teaching and learning process is investigated. The aim of this paper thus, is to further examine the combination of animation, text and audio to help novice students to better understand basic concepts of object-oriented programming.

Visual stimuli in form of solely animation with and without auditory stimuli are examined in this paper. Visual stimuli in form of animation and text with and without auditory stimuli are also examined in this paper.

A set of studies was conducted for first year computer science students in object-oriented programming. In the studies, animation, text and audio were combined in different combinations in software visualization prototypes. The software prototypes described in prototype design were exposed to the students and to evaluate the perceived usefulness of the prototypes an exam was carried out. The examination consisted of a questionnaire to test the acquired knowledge of the utilization of the prototypes. To collect the students thoughts and

978-1-4799-7732-1/14/$31.00 ©2014 IEEE

feelings and the perceived usefulness of the prototypes interviews was also conducted.

II. Theoretical Base

This section discusses software visualization concepts and describes the underlying theories of this paper; dual-coding theory and epistemic fidelity theory.

Software visualization

Software visualization is static or animated visual representations of information about software systems. In this study, software visualization is referred to as graphical representations in the form of animations of programs and algorithms.

Further, software visualizations can be declared as a context of communication between the designer of the visualization and the user of the visualization. And it is associated with principles from cognitive psychology and perception [2]. Program visualization and algorithm visualization are two of the underlying concepts of software visualization. Their definitions are outlined as follows:

- *Program visualization* – programs are mapped to be represented graphically [3]. Thus, different states of the program or a running program are illustrated graphically, while the program is defined textually, for example in form of source code [4].
- *Algorithm visualization* – a high-level description of a piece of software is visualized [5], not related to the program's source code and without details [6], often used for a better understanding of the procedural behaviours of the algorithms [7].

Dual-Coding Theory (DCT)

Dual-coding begins with the assumption that the activity of two distinct symbolic systems to a large extent are the constituents of cognition. The first symbolic system encodes non-verbal events, for example animation only or animation in combination with text. The other symbolic system encodes verbal events. The systems are functionally independent but partly interconnected.

According to the dual-code hypothesis, viewers of a visualization who encode knowledge in both non-verbal and verbal modes are able to build dual representations in the brain and between those representations, referential connections [7].

For example, if a student is asked to remember the sentence "*the children played outside*", it would be much easier to remember it if the student had a mental picture or activity that accompanied the verbal information. A picture that shows children playing outside would probably be ideal. In that case, the student would have a mental model[1] corresponding with the verbal code [8]. Software visualizations that apply dual-coding make the transfer of target knowledge more efficient. [7].

Epistemic Fidelity Theory (EFT)

Epistemic fidelity theory assumes that symbolic models of the physical world carried around in people's minds are the basis of their reasoning and action [7].

According to the theory, knowledge can be instantiated in humans' minds as symbolic structures but exists independently of humans[9]. Further, in accordance with the theory, graphical representations have a large capacity to support representations that match an expert's mental model of an algorithm closely.

Thus, EFT stresses the value of a good match between the mental model of the expert and the graphical representation. The more robust and efficient the transformation of the mental model is to the viewer of the visualization, the higher the match fidelity. This means that the viewer of the visualization is able to decode and internalize the target knowledge better [7].

A version of EFT can be regarded as *strong* if it asserts that the transfer of knowledge occurs solely by taking the epistemic fidelity into account, not other factors [9]. On the contrary, a *weak* version of ETF holds that certain characteristics beyond and above epistemic fidelity, play an important role in successful transmission of knowledge, while the same underlying epistemology is intact [9]. One example of a weak version of the ETF is the formerly presented DCT.

III. METHOD

This section describes the methods of how to study how effective the combination of software visualization, in form of animation, text and audio are.

[1] A mental model refers to a description of a maintainer's mental representation of what is to be understood [10].

Since this study as a whole consists of both algorithm visualization and program visualization, there are two main studies carried out: Study 1 concerning algorithm visualization and Study 2 concerning program visualization.

First, the aim of each study is presented, then the working method is described, i.e. the set-up, the data gathering and the data analysis.

Study 1

The aim of this study is to test the effectiveness, or the perceived utility of an *algorithm visualization* prototype with and without verbal stimuli, to help novice students to understand central concepts of object-oriented programming. The algorithm visualization is composed of both animation and text, together with auditory stimuli.

An algorithm visualization prototype was prepared. The prototype illustrates an evolving process of the running algorithm. The illustrated process merges gradually into a depicted result of the algorithm. The prototype is composed of animation, text and the recording of an expert's oral explanation of the process (called Prototype 1a). As a reference, the same algorithm visualization prototype was used but without auditory stimuli (called Prototype 1b).

An experiment was conducted to carry out Study 1. Participants of the experiment were nineteen (19) voluntary computer science students at the Department of Computer and Systems Sciences, Stockholm University. The participants had at least some former knowledge of Java, i.e. the participants had been obligated to attend at least one former introductory Java programming course. Nine (9) of the respondents were shown prototype 1a, while the other half, (10) respondents were shown prototype 1b. A written exam in form of a questionnaire was completed by the participants directly following the demonstrations.

In order to gain deeper knowledge concerning the algorithm visualization prototypes, three voluntary participants from the above group were interviewed. The questions of the interviews were a set of fixed questions.

Study 2

The aim of this study was to test the effectiveness, or the perceived utility of a program visualization prototype with and without verbal stimuli, to help novice students to understand central concepts of object-oriented programming.

A program visualization prototype was prepared. The prototype illustrates an object-oriented view and the dynamics and manipulation of the objects when running a software program. The illustrated process of the program merges gradually into a depicted result of the program. The prototype is composed of animation and a recording of an expert's oral explanation of the illustrated process (called Prototype 2a). As a reference, the same program visualization prototype was used but without auditory stimuli (called Prototype 2b).

An experiment was conducted to carry out study 2. Participants of the experiment were nineteen (19) students with the same prerequisites as the participants in Study 1. The participants were presented both Prototype 2a and Prototype 2b. Nine (9) of the respondents were shown prototype 2a, while ten (10) respondents were shown prototype 2b. The same exam, i.e. questionnaire as in Study 1 was completed by the participants directly after the demonstrations.

Similarly to Study 1, three (3) voluntary participants from the above presented group were interviewed. The questions of the interviews were the same set of fixed questions as in Study 1.

IV. PROTOTYPE DESIGN

This section describes the prototypes used in the data gathering part. The algorithm visualization prototype and the program visualization prototype in the studies were designed and developed by the author. Subsequently, auditory stimuli, an expert's oral explanation of the processes of the prototypes, have been added. Both prototypes will be explained in detail in the following.

Algorithm visualization prototype

The algorithm visualization prototype begins with the display of the class, in which the algorithm is found. The algorithm used is a "for loop" statement that represents iteration. The declaration of variable *x* in the class For_example that is used in the "for-loop" statement is then illustrated. Further, the algorithm visualization proceeds with the visualized execution of the "for-loop" statement, i.e. the operations of the "for-loop" statement and the order in which the operations are performed. An illustration of the gradually emerging result of the executing "for-loop" statement in the Command Prompt is also illustrated.

The algorithm is gradually visualized from plain program code, to the illustration of the code, to the transformation of the illustrated code, into the illustration of the result of the algorithm, in the Command Prompt.

The window consists of two frames. First, there is one minor frame in the upper left corner. In this frame, the program code, which contains the "for-loop" statement is displayed. Second, there is one large frame in the centre, aligned to the right. In this frame, the illustration of the "for-loop" statement being processed is presented. The gradually emerg-

ing result of the "for-loop" statement is displayed in the lower left corner.

A coloured square which corresponds to the UML notation is used, to illustrate the class in which the "for-loop" statement is found.

Variables are represented as boxes that hold the values of the variables. The boxes have the same colours as the objects in which they are found. The names of the variables are placed on the left side of the boxes.

The algorithm visualization prototype is complemented with textual clarifications of different states in the algorithm visualization. The textual clarifications are presented in form of small commentary text areas, being visible as the visualization proceeds. The contours of the text areas and the clarification text written inside the text areas are red.

To follow each round (iteration) of the "for-loop" statement, the number of each round, is displayed. The display of each round is placed outside the rectangle that contains the "for-loop" statement, on the left side. The colour of the display, i.e. the numbers of each round is red.

The program code in the upper left frame is highlighted with a yellow translucent colour. The highlighting starts from the beginning of the code, row by row, until the end of the code. The parts of the program code that is highlighted for the moment, is the part that is visualized in the large frame in the centre. This is to facilitate the focus of what in the code is being visualized at a certain moment.

The iterative process of the actual algorithm is highlighted as well. The highlighting illustrates the operations of the "for-loop" statement and the order in which the operations are performed, when executing the "for-loop" statement.

Two snapshots at different states of the algorithm visualization prototype are presented in Fig. 1 and Fig. 2 as follows:

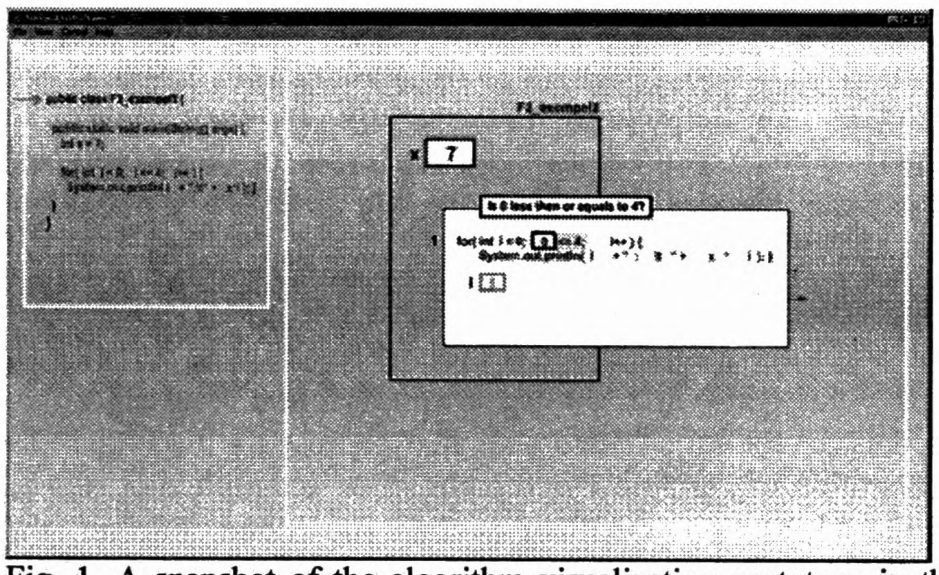

Fig. 1. A snapshot of the algorithm visualization prototype in the beginning of its process.

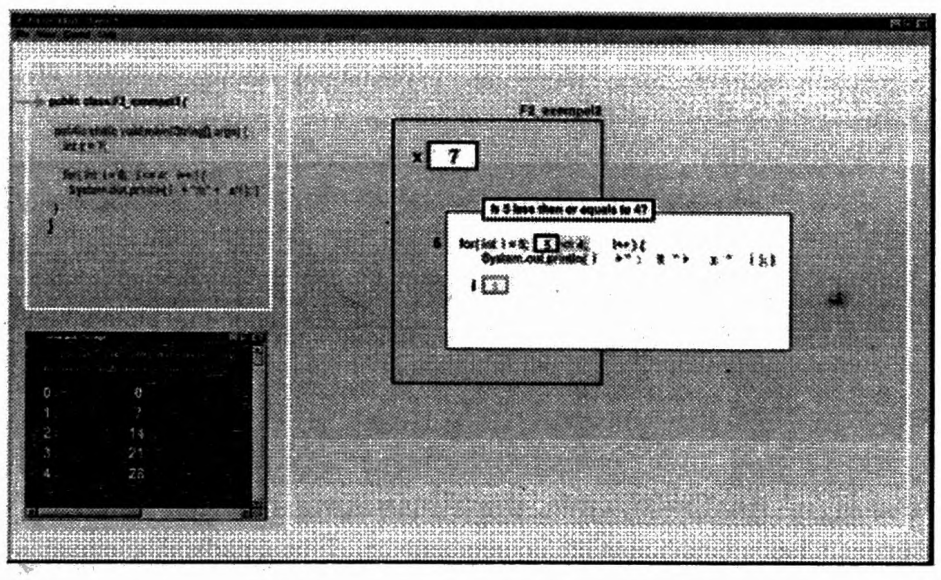

Fig. 2. A snapshot of the algorithm visualization prototype in the end of its process.

Program visualization prototype

To illustrate the object-oriented view, the program visualization prototype begins with a display of a core outline of the classes. This is to scope the overall picture of the program's complexity. The program visualization then proceeds to illustrate the dynamics of the running program. In addition, an illustration of the gradually emerging result of the running program on the screen, i.e. the Graphical User Interface, is illustrated.

Thus, the program is visualized from plain program code, to the visualization of the code, to the transformation of the visualization of the code, into the visualization of the outcome of the program on the screen.

The window is composed of two frames. First, there is one minor frame in the upper left corner. In this frame, the program code is displayed. Second, there is one large frame in the centre, aligned to the right. In this frame, the illustration of the running program is presented. The gradually emerging result of the program is displayed in the lower left corner.

Coloured squares corresponding to the UML notation is used, to illustrate classes in Java. Different tones of the colours are used to indicate correlations between classes and objects. The names of the classes and the objects are placed above the squares, representing classes and objects, in the upper right corner.

The representations of the standard classes are coloured grey. Further, they are combined with unfilled arrows, to illustrate Java's inheritance structure. This is also consistent with the UML standard concerning inheritance.

Reference variables are represented as boxes holding the references. The boxes have the same colours as the objects in which they are found. The names of the reference variables are placed on the left side of the boxes.

Declarations of reference variables are illustrated as growing arrows. The arrows have the same colours as the classes which contain the reference variables. Further, the arrows are expanding from the inside of the variable boxes, until they reach the instantiated objects.

When an object is instantiated, the colour of the origin square, representing the class of which the instantiated object belongs to, becomes intensified. This illustrates that the square is now representing the instantiated object.

Methods and constructors of classes and objects, which are not standard classes in Java, are illustrated as solid lined ovals on the left border of those classes and objects. In addition, constructors are filled with yellow colour. Methods of classes and objects, which are standard classes in Java, are illustrated as broken lined ovals.

When a method is called, the contour of the oval, representing the method, becomes coloured like the object in which the method is found. The contour of the oval also becomes solid if the method is located in a standard class. When a method call finished executing, the line of the oval representing the method, remains solid and the colour of the line is diminished, to illustrate the executed method call.

The program code in the upper left frame is highlighted with a yellow translucent colour. The highlighting starts from the beginning of the code and proceeds until the end of the code. The parts of the program code that is highlighted for the moment, is the part that is visualized in the large frame in

the centre. This is to facilitate the focus of what in the code is being visualized at a certain moment.

Two snapshots at different states of the program visualization prototype are presented in Fig. 3 and Fig. 4 as follows:

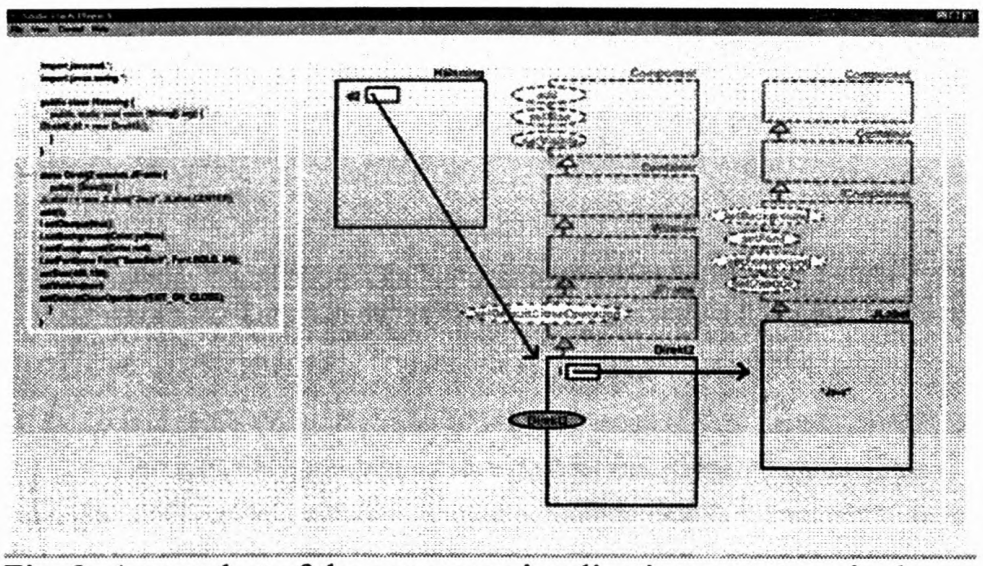

Fig. 3. A snapshot of the program visualization prototype in the middle of its process.

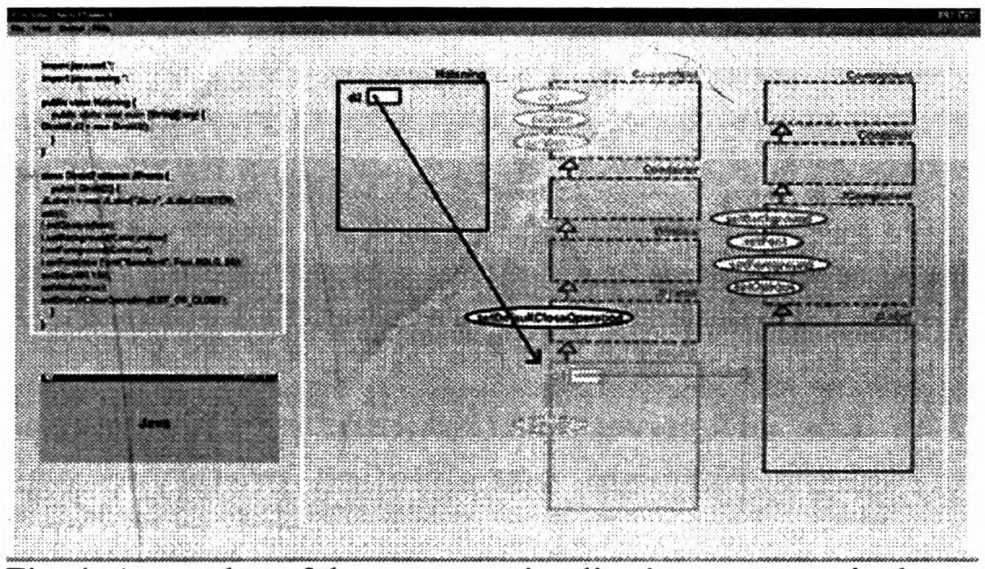

Fig. 4. A snapshot of the program visualization prototype in the end of its process.

V. RESULTS

This section describes the results of the studies. The results are a set of observations on the effectiveness, or the perceived utility of combining animation, text and audio based on the results of the exam and on the interviews. The algorithm visualization prototypes utilized in Study 1 will be referred to as follows:

- **AV(Y)** – Algorithm Visualization (Auditory stimuli –Yes), i.e. an algorithm visualization prototype composed of both animation and text as the visual stimuli, together with auditory stimuli. This was called Prototype 1a in section 3.
- **AV(N)** – Algorithm Visualization (Auditory stimuli –No), i.e. an algorithm visualization prototype composed of both animation and text as the visual stimuli but without auditory stimuli. This was called Prototype 1b in section 3.

Likewise, the program visualization prototypes used in Study 2 will be referred to as:

- **PV(Y)** – Program Visualization (Auditory stimuli –Yes), i.e. a program visualization prototype with visual stimuli in form of animation only, together with auditory stimuli. This was called Prototype 2a in section 3.
- **PV(N)** – Program Visualization (Auditory stimuli –No), i.e. a program visualization prototype with visual stimuli in form of animation only, and without auditory stimuli. This was called Prototype 2b in section 3.

Results of Study 1

The results from the experiments and exam/questionnaires are specified in the form of observations.

1.1. The effectiveness (in the form of comprehension) of an algorithm visualization prototype using text and animation in combination was *not* enhanced by adding auditory stimuli to the prototype. The students' comprehensions of the algorithm were not enhanced by the auditory stimuli. The text presented in the algorithm visualization prototypes may have prevented the effectiveness of the auditory stimuli of the first prototype.

1.2: The algorithm visualization prototypes provided a clear focus throughout the whole visualization. 8 out of 9 of the participants that were presented the algorithm visualization prototype being composed of animation and text, together with auditory stimuli, thought that the focus was clear throughout the whole algorithm visualization. Furthermore, 9 out of 10 of the participants that were presented the algorithm visualization prototype being composed of animation and text, without auditory stimuli, also thought that the focus was clear throughout the whole algorithm visualization. The presumed reason for the majority's impression that the focuses of the prototypes were clear, could be that the prototypes are sequentially formed and iterative. This could have facilitated the understanding of the "for-loop" statement. However, the auditory stimuli of the first algorithm visualization prototype did not enhance the effectiveness (in the form of focus) of the visualization.

1.3. Both of the algorithm visualization prototypes used are considered to be a good way of transferring knowledge about the algorithm. 7 out of 9 of the participants that were presented the algorithm visualization prototype consisting of ani-

mation and text, together with auditory stimuli, thought that the visualization was a good way of transferring knowledge about the algorithm. Further, 9 out of 10 of the participants that were presented the algorithm visualization prototype consisting of animation and text, without auditory stimuli, also thought that the visualization was a good way of transferring knowledge about the algorithm.

Consistent with the discussion on observation 1.2, the supposed reason for the majority's positive attitudes towards the prototypes capacity of transferring knowledge about the "for-loop" statement, is that the prototypes are sequentially formed and iterative, which facilitated the understanding of the "for-loop" statement. Yet, no noticeable differences could be measured by the added auditory stimuli of the first algorithm visualization prototype. That is, whether or not the students considered the algorithm visualization prototypes to be a good way of transferring knowledge.

To gain a deeper understanding, interviews were conducted and the observations made are the following.

1.4. The AV(Y) can facilitate the learning of a "for-loop" statement and its processes. The AV(Y) iterates a certain number of rounds, controlled where to focus, which enables time for reflection and an understanding of how the "for-loop" statement is proceeding. In addition, the animation of the "for-loop" statement is clear and distinct and indicates to be a well-adjusted way to explain a "for-loop" statement for a novice-level Java student.

1.5. The AV(Y) can facilitate the learning of variables. The animation of variables and variable boxes clearly illustrate the change of values of the variables, which otherwise can be arduous to picture.

1.6. The AV(Y) was perceived to be prolonged and slow. Due to the iteration of the number of rounds of the "for-loop" statement, the AV(Y) appears to be tedious for someone that already understands the processes of a "for-loop" statement or understands the principle of a "for-loop" statement fast. This is likely due to the implementation of the prototype rather than an inherent problem of this kind of visualization.

1.7. The AV(N) can likewise facilitate the learning of a "for-loop" statement and its processes. The animation of the "for-loop" statement is described as being exhaustive, clear and descriptive and to be having the capacity of knowledge transfer. In addition, the AV(N) were expressed to indeed picture the constitution of a "for-loop" statement and corresponding to the AV(Y) it is experienced to provide a distinct focus throughout the whole animation. It indicates to be a suitable method to teach the evolving process of a "for-loop" statement for novice-level Java students. This is due to the iteration of the exact process of every round of the "for-loop" statement. The "for-loop" statement's long iterative process of five rounds, allowing time for reflection and understanding were appreciated.

1.8. The AV(N) was also perceived to be slow. As in observation 1.6, the animation of the "for-loop" statement was experienced to be too slow for students who had experience of Java. The conclusion is the same as for that observation.

Results of Study 2

The results from the experiments and exam/questionnaires are again specified in the form of observations.

2.1. The effectiveness (in form of comprehension) of the program visualization prototype was *not* enhanced by adding auditory stimuli to the prototype. The program visualization prototypes used in Study 2 where not addressed any noticeable differences when evaluated from the questionnaires. That is, the students comprehension of the program, were not enhanced by the auditory stimuli. The result is *not* in line with the dual coding theory (DCT).

2.2. The focus of the program visualization prototype was enhanced by adding auditory stimuli to the prototype. 7 out of 9 of the participants that were presented the program visualization prototype being composed of animation together with auditory stimuli, thought that the focus was clear throughout the whole program visualization. Only 3 of 10 of the participants that were presented the program visualization prototype being composed of animation without auditory stimuli, thought that the focus was clear throughout the whole program visualization. In line with DCT, the auditory stimuli did add further value to the visualization, in form of increased focus.

2.3. Both of the program visualization prototypes are considered to be a good way of transferring knowledge. 7 out of 9 of the participants that were presented with the program visualization prototype

consisting of animation together with auditory stimuli thought that the program visualization was a good way of transferring knowledge about the program. Further, 8 out of 10 of the participants that were presented the program visualization prototype consisting of animation without auditory stimuli, thought that the program visualization was a good way of transferring knowledge about the program. Yet, no noticeable differences could be measured by the added auditory stimuli of the first program visualization prototype. That is, whether or not the students considered the program visualization prototypes to be a good way of transferring knowledge.

To gain a deeper understanding, interviews were conducted and the further observations made are the following.

2.4. PV(Y) can facilitate the learning of basic concepts of object-oriented programming. The PV(Y) indicates to facilitate learning of the object-oriented view, inclusive classes vs. objects, reference variables, method calls etc. However, this appears to be applicable mostly to students with former experience of Java and the subsets stated above. Yet, the supplementary recording of an expert's oral explanation of the illustrated process of the running program, appeared to facilitate comprehension. Further, highlighting of the program code to facilitate focus of the illustrated subsets of the code and the narrative exposure of the main method were appreciated.

2.5. The PV(Y) contains too much simultaneously displayed information. The simultaneously displayed information was experienced to be confusing for the students. In addition, the animation is perceived to be too fast in speed. The PV(Y)'s information overload and rapid speed results in student's loss of focus and difficulties to match program code with animation. This is likely due to the implementation of the prototype rather than an inherent problem of this kind of visualization.

2.6. PV(N) can likewise facilitate the learning of basic concepts of object-oriented programming. The PV(N) indicates to facilitate learning of the object-oriented view, inclusive classes vs. objects, reference variables, method calls etc. Similarly to PV(Y) it appears to be foremost applicable to students with experience of Java and the subsets defined above. Students with experience of Java also had a greater ability to focus on the executing program being illustrated.

2.7. The PV(N) also contains too much simultaneously displayed information. Asserted by the participants being exposed to the PV(N), it was corresponding to the PV(Y), arduous to keep focus on the illustrated process of the PV(N) for novice-level Java students, due to relatively high speed and simultaneous information on the screen. The PV(N) were expressed to be more indistinct than the AV(N) and equivalent to the PV(Y), the animation of the program is found to be proceeding too fast. A mapping between the program code and the visualization needs to be done and a demand for the PV(N) to be explained were requested.

VI. CONCLUSIONS

In general, the findings of the study support visualization as an aid in teaching and learning programming. But the results on the additional use of audio as a complement to text and animation varies. The effectiveness of the algorithm visualization prototypes in form of comprehension were not enhanced by adding auditory stimuli to the prototypes. This can be explained in two ways. Either the algorithm used in the experiment was not complex enough, or understanding algorithms (a higher-level task than understanding code execution) is not facilitated as much by added audio. The auditory stimuli of the program visualization, on the other hand, did add further value to the visualization, in form of increased focus. The results are partly in line with DCT since attention (focus) but not comprehension was enhanced.

VII. SUMMARY

The aim of this paper was to examine how effective a combination of animation, text and audio are in order to help novice students to better understand basic concepts of object-oriented programming.

The overall method used in this paper, was to carry out a set of studies on first-year computer science students in object-oriented programming, in which animation, text and audio were, and were not, combined, in software visualization prototypes. In order to measure how effective the software prototypes were, the students were carried out an exam in form of a questionnaire. It was to test the acquired knowledge from the teaching and learning process where the software visualization prototypes were used. Interviews with the students were

also carried out in order to gather the students' experiences of the software visualization prototypes.

The result of this paper is a set of observations about the effectiveness of combining animation, text and audio, based on the exam/questionnaire carried out by the students, and a set of experiences, in form of opportunities and problems, gathered from interviews of the students, after using the software visualization prototypes.

VIII. FUTURE WORK

The fact that the participants of the study had at least some former knowledge of Java, could mean that the participants' equal levels of comprehension was due to former knowledge.

Further work should therefore include participants with no former programming knowledge. It is very interesting in a follow-up study to investigate why DCT could only explain the attention outcome and not the comprehension of the present study. This should form a basis for a more detailed understanding of how to employ visualization in real classroom settings.

ACKNOWLEDGMENTS

I would like to thank Peter Mozelius at the Department of Computer and Systems Sciences, Stockholm University, for the voice recording, and adding audio to the software visualization prototypes used in this study.

REFERENCES

[1] Eckerdal, A. 2009. Novice Programming Students' Learning of Concepts and Practise. A Dissertation. Department of Information Technology, University of Uppsala, Sweden.

[2] Petre, M., 1995. Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming. ACM, New York.

[3] Roman, G-C., Cox, K, C,. 1993. A Taxonomy of Program Visualization Systems. IEEE Computer.

[4] Myers, B, A., 1986. Visual Programming, Programming by Example, and Program Visualization: A Taxonomy. CHI'86 Conference Proceedings (59-66). ACM, New York.

[5] Price, B, A., Baecker, R, M., Small, I, S., 1993. A Pincipled Taxonomy of Software Visualization. Journal of Visual Languages and Computing 4(3): 211-266.

[6] Myller, N., 2004. The Fundamental Design Issues of Jeliot 3. Master's Thesis. Department of Computer Science, University of Joensuu.

[7] Hundhausen, C, D., Douglas, S, A., Stasko, J, T., 2002. A Meta-Study of Algorithm visualization Effectiveness. Journal of Visual Languages and Computing (2002) 13, 259-290.

[8] Gallegos-Butters, A, M., Schneider, G, P., 2004. Educational Multimedia: A test of Dual-Coding Theory. University of San Diego.

[9] Hundhausen, C, D., 1999. Toward Effective Algorithm Visualization Artifacts: Designing For Participation and Communication in An Undergraduate Algorithms Course. A Dissertation. Department of the Computer and Information Science, University of Oregon.

[10] Storey, M.-A, D., Fracchia, F, D, Müller, H, A,. Cognitive Design Elements to Support the Construction of a Mental Model during Software Exploration.