Adaptive, Engaging, and Explanatory Visualization in a C Programming Course

Peter Brusilovsky and Michael Spring
School of Information Sciences
University of Pittsburgh
Pittsburgh PA 15260
{peterb, spring} @ mail.sis.pitt.edu

Abstract: This paper discusses three ways to improve the educational value of program visualization: *engaging* visualization, *explanatory* visualization, and *adaptive* visualization. We present some tools that we have developed to explore adaptive, engaging, and explanatory visualization in the classroom: a system for exploring the calculation of C expressions (WADEIn) and a set of small learning environments for exploring some of the "muddiest points" of C. We also provide some results of our most recent study of the WADEIn system in the classroom and summarize the directions for future work.

Introduction

Interactive visualization is a powerful educational tool. Visualization can provide a clear visual metaphor for understanding complicated concepts and uncovering the dynamics of important processes that are usually hidden from the student's eye. Visualization has been used to enhance the teaching of various subjects ranging from chemistry (Yaron et al., 2001) to mechanics (Hampel, Keil-Slawik & Ferber, 1999) to physics (McKenna & Agogino, 1997). One of the most active application areas for educational visualization research is visualization of program execution in the context of Computer and Information Science (CIS) education. Program visualization is considered one of the most powerful educational tools in CIS education. Visualization has been explored in the context of machine-level languages (Butler & Brockman, 2001), various high-level languages (Domingue & Mulholland, 1998; Haajanen et al., 1997; Tung, 1998), and algorithms and data structures (Brown & Najork, 1996).

For many years, the focus of visualization research was on developing better tools and trying visualization in various contexts. Few studies of the effectiveness of visualization were done since the benefits of visualization appeared evident for many researchers and teachers. Yet, It has been shown in several experiments (Byrne, Catarambone & Stasko, 1999; Stasko, Badre & Lewis, 1993) that the educational effect of observing visualizations is unexpectedly low. Often students failed to understand what was happening inside a program or an algorithm in the presence of a well-developed visualization. Several researchers and teams has focused their efforts on "making visualization work". We see three promising approaches to "useful visualization" that can be called *engaging* visualization, *explanatory* visualization, and *adaptive* visualization.

Proponents of *engaging visualization* suggest that the major problem of traditional visualization is its passive nature. In traditional visualization, the role of the student is simply to watch what's happening on the screen. The only kind of action that is allowed (and not even in every system) is to control the pace of visualization. Several psychological theories used in education - from activity theory to constructivism stress that a student has to be actively engaged in the educational process for the learning to happen. Guided by this prospect several projects have attempted to build some more engaging visualization and to measure its efficiency. Several ways were suggested to engage the students and to make them work more actively with the visualization. These techniques range from "light activity" such as developing personal data sets for visualization to "heavy activity" such as asking the student to construct the whole visualization instead of watching a "prepared" one. These techniques brought positive results (Hundhausen, Douglas & Stasko, 2002). From our point of view, the most attractive way to engage the student is to ask her to predict the results of animation. This way has been shown to be effective (Byrne, Catarambone & Stasko, 1999), it is easy to implement, and it allows visualization to be used not only for presentation but also for evaluation.

Proponents of *explanatory visualization* argue that the students often fail to learn from visualization because they can't understand what they see - what has happened, why it has happened and how the particular behavior connects to general knowledge presented in class (Brusilovsky, 1994). The idea of explanatory visualization is to augment every animated step of visualization with natural language explanations. The role of the

explanations is to tell what is going on, why it happens, and how it relates to general principles. It has been shown that explanations indeed help the student to understand what they see (Brusilovsky, 1994). The programming "problets" developed by Amruth Kumar (Kumar, 2001; Shah & Kumar, 2002) provide very impressive examples of explanatory visualization. *Adaptive visualization* is based on an assumption that a student may have different level of knowledge of different *elements* of a program or an algorithm that is being visualized. In the case of a program, the student may know some high-level language constructs or machine level language commands better than others. For the case of algorithm animation the student may understand some steps of an algorithm better than others. In this context, regular visualization that animates all constructs or steps for each student with the same level of details may be the wrong approach. For a troublesome construct the level of detail may not be deep enough for the student to understand its behavior. A visualization of a well-understood construct with unnecessary details may distract the student and make it harder to focus on and thus comprehend the behavior of the constructs that are still poorly understood. The idea of adaptive visualization is to match the level of detail in the visualization of each construct or step to the level of student knowledge about it. The less the level of understanding of a construct, the greater the level of detail in the visualization. Naturally, with a demonstrated increase in student knowledge about specific constructs, the level of visualization of those constructs should decrease. This approach allows a student to focus attention on the least understood components while still being able to understand the whole visualization. An experimental evaluation of adaptive visualization in the context of program debugging has confirmed that adaptive visualization can improve student learning (Brusilovsky, 1993).

We have explored all three ways of improving the efficiency of visualization listed above focusing most on adaptive and explanatory visualization (Brusilovsky, 1992; Brusilovsky, 1993; Brusilovsky, 1994). The work has been performed in a rather simple context of an educational mini-language, Tortoise (Brusilovsky et al., 1997). Our current challenge is to explore the scalability of engaging, explanatory, and adaptive visualization in a more challenging and more practical context of a C programming course. We have developed several Web-based tools that offer engaging, explanatory, and adaptive visualization of various aspects of C program execution. We have used these tools in several undergraduate courses and conducted studies of the impact. This paper presents some of our tools: a system for exploring the calculation of C expressions (WADEIn) and a set of small learning environments for exploring some of the "muddiest points" of C. We also provide some results of or most recent study of the WADEIn system in the classroom and summarize the directions of the future work.

WADEIn: An Adaptive Visualization of Expression Evaluation

For the students in our programming and data structure courses based on C language, expression evaluation is one of the most difficult concepts to understand. They have problems with both understanding the order of operator execution in a C expression and understanding the semantics of operators. To help the students, we have developed a Web-based Adaptive Expression Interpreter (WADEIn, pronounced wade-in). WADEIn allows the student to explore the process of expression evaluation step-by-step with detailed animation as well to check their understanding of the order and the results of evaluation. From a research point of view, the goal of WADEIn is to explore adaptive and engaging visualization. At the moment, WADEIn does not use explanatory visualization. Since this paper is focused on what an effective visualization can be, we present the WADEIn system from a user point of view. Technical details such as student modeling mechanism, the architecture, and the implementation can be found in a paper on the earlier version of the system.

The core of WADEIn is an expression interpreter that can work in either exploration or evaluation mode. In *exploration mode* the user can observe the process of evaluating a C expression step-by-step. An expression can be typed in or one can be selected from a menu of expressions. At the beginning of evaluation, the system indicates the order in which various operations in the expression will be performed (Fig. 1). After that, the system starts visualizing the execution of each operation. The goal here is to show the results and the process of executing an operator. To show the results, the system visualizes a "shrinking" copy of the original expression and the values of all involved variables.

The execution of every operation is split into several sub-steps. First, the system highlights the operations to be executed and its operands on the "shrinking" copy of the expression and re-writes them to the *evaluation area* field (gray rectangle in the center of the window). Next, it shows the *value* of the expression (2 on Fig. 1). In case of assignment and increment/decrement operations, WADEIn also shows the new value of the variable involved (Fig. 1). In the next sub-step, it replaces the whole highlighted operation in the "shrinking" copy with the calculated value. If a variable has changed its value as a side effect of evaluation, it also changes the value of the variable (Fig. 2). On this sub-step the system may use animation by "flying" the numbers from the working field to their destinations in an expression or in the variable area. Finally, the system removes all highlighting, "shrinks" the simplified expression, and prepares for the next step.

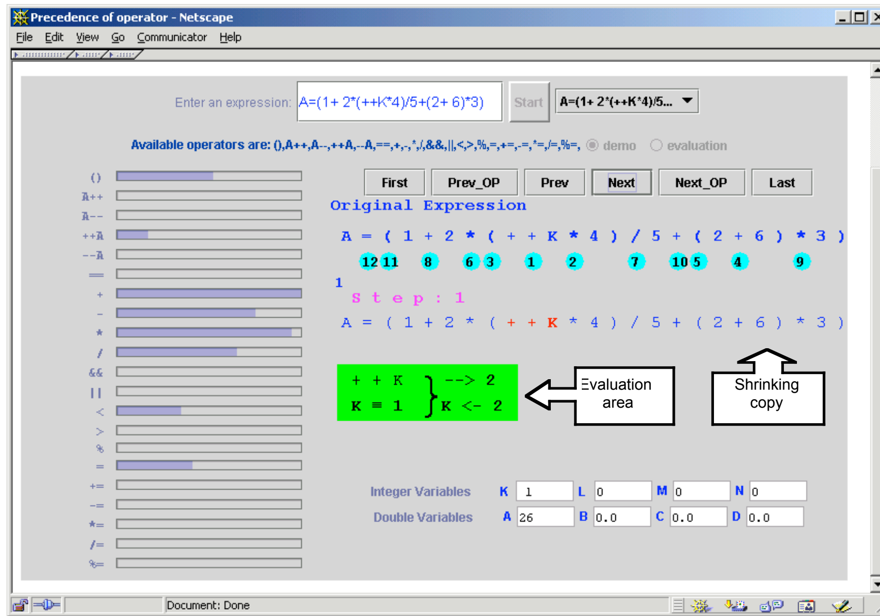

Figure 1. The user starts working with an expression. Numbers in circles show the order of calculation. The applet starts visualizing the execution of the first operation ++K (the current value of variable K shown below is 1).

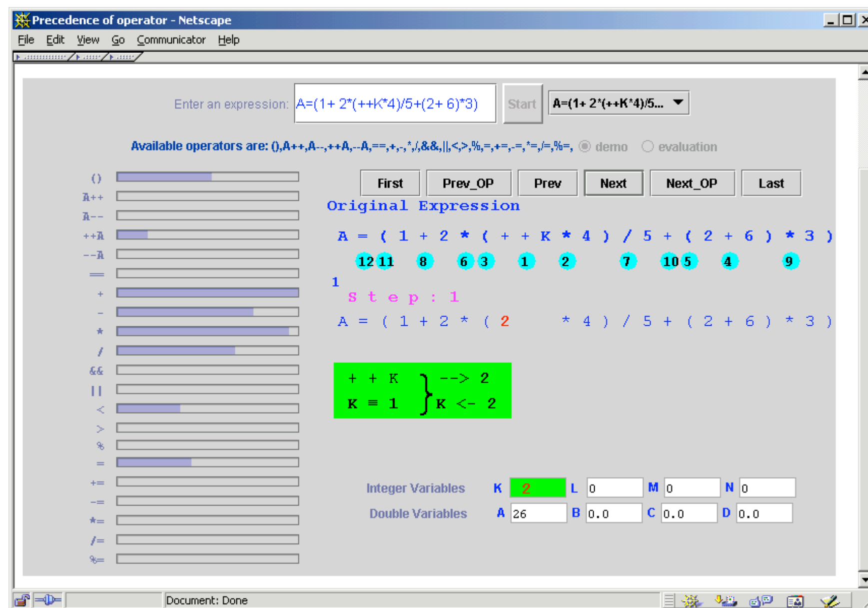

Figure 2. The results of calculating an operation are "flying in" to replace the original operation and operands in a working expression and the old value of the variable involved.

The level of detail in executing an operation depends on the user's current level of knowledge. For the minimal level of knowledge (1.0), the system will perform all sub-steps and will show the animation in slow motion. As the user learns an operation and his or her knowledge level improves from 1 to 5, the system gracefully degrades the level of detail in the visualization by increasing the speed of animation and removing some sub-steps. For the highest level of knowledge (5.0) there are no sub-steps and no animation - the operation is executed in one step. To make the adaptive behavior of visualization more transparent, we choose to visualize the system's opinion about student knowledge with progress indicator bars. Students have expressed a clear preference for the progress bars as a way to understand the work done and the work left to be done.

To control the process of expression interpretation, the user has six buttons. *First* and *Last* let the user move to the beginning or the end of expression execution; *Next_OP* and *Prev_OP* let the user move in one step to the beginning of interpretation of the next or previous operation; *Next* and *Prev* move the user one adaptive step forward and backwards. Normally, the user would use only *Next* button (or simply hit the return key). Other buttons can be used to watch the same operations and sub-steps again and again forwards and backwards or to skip sub-steps or operations. This variety of ways to control the evaluation supports the multiple uses of WADEIn. The primary use of WADEIn is for the students to explore in detail the newly presented operators. This is exactly the context that the adaptive *Next* and *Prev* buttons support. It is known that the students who are new to the environment and the domain prefer to use a single button to navigate through learning material. Yet we also anticipate more skilled users who need more control over the execution such as teachers using WADEIn to present new operators in class or students returning to WADEIn for quick targeted experiments in the process of problem solving or exam preparation.

To make student work in exploratory mode more engaging and active, we allow the student to provide both the expressions and the data for the evaluation process. While a good set of interesting expressions to explore can be provided by a teacher (these expressions are available through the pop-up menu to the right of the Start button), the student can enter and visualize any syntactically correct expression. In addition, before the start of visualization, students can provide their own values for all the variables used by the system. As we have mentioned above, this form of engaging is known to be efficient.

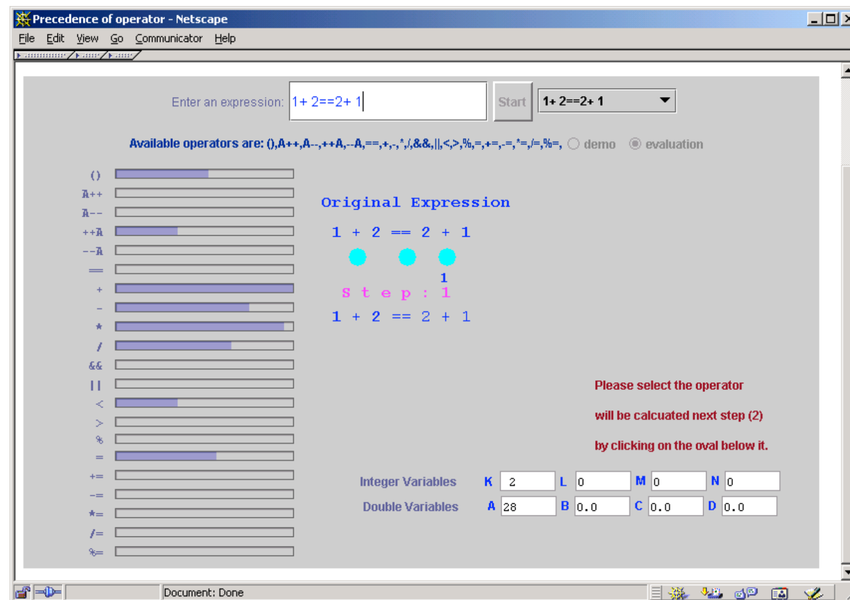

Figure 3. At the beginning of executing an expression in the evaluation mode, the system requests the user to mark the order in which the operators will be executed.

The *evaluation mode* provides another way to engage students as well as to evaluate their knowledge. Here the student has to do most of the work that is done by the system in the exploration mode. In the evaluation mode the student starts working with the new expression by indicating the order of execution of the operations in the expression (Fig. 3). After that, the system corrects errors, shows the correct order of execution, and starts evaluating the expression (Fig. 4). The interface in the evaluation mode is quite similar to the one of exploration mode - the execution of every operation is adaptively visualized. However, if the student knowledge of the current operation is lower than a predefined threshold, the system will not calculate the result of the operation, but instead challenge the student to provide the result (Fig. 4). If the user makes an error, the correct result is provided, so the calculation of the expression is always correct. To keep the student engaged and challenged, the well-understood operations are performed by the system in exploration mode. To allow the use of evaluation mode for exercises and knowledge evaluation, student freedom of navigation through the solution is limited. Only two actions are possible - quit an exercise and move to the next operator.

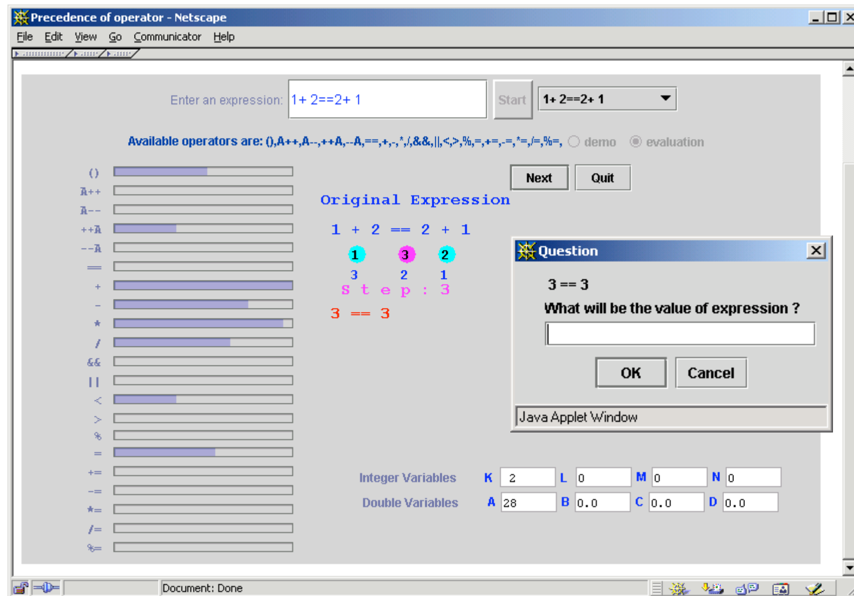

Figure 4. In evaluation mode, the system requests the result of each operation from the user if his or her level of knowledge about this operation is low.

Explanatory Visualization of the "Muddiest Points" of C Language

The C language is not designed as a language for teaching programming, but it is used heavily in a variety of programming courses including introductory programming and data structures. Every teacher dealing with C or C++ in class is well aware of some of the "muddy points" of the C language that are known to create problems for nearly every student. Such topics as buffer overflow, pointer assignment, pointer arithmetic's are hard to explain and hard to understand. The mastery of these topics is critical for student success. To help teachers to explain these topics and students to comprehend them we have developed a set of learning environments for the "muddiest points" of C.

Our environments use the idea of explanatory visualization. Each environment presents visually one or more problem situations - small programs that demonstrate some aspect of the C language. On the screen the student can see the program code and a visual representation of data. They are designed to help the student understand the concept being demonstrated. Working with problem situations, the student can navigate through the execution of the target program forwards and backwards. As in many other program visualization environments, each step of the execution is visualized - i.e., the environment dynamically shows all changes in data introduced by the current step and any output generated by the step. Output is presented in a "console" window located under the code of the program (Figures 4 and 5). In addition, each step is followed by a natural language explanation of what has happened, why it has happened, what hidden problems are demonstrated and which lessons can be learned from the step. These explanations mimic narration that a teacher usually presents when demonstrating program execution in the classroom. It is these explanations that we consider critical for understanding program visualization.

The presented environments constitute our first attempt to use explanatory visualization in the context of teaching C. Our priority was not to "make it right" from the first attempt, but to create a useful tool that we can immediately use in the classroom for teaching and research. So far they have already been used in one class of about 30 students. Our current focus is on making these environments more engaging, allowing the students to do a bit more than just playing the visualization forwards and backwards. For example, some environments (Figure 5) allow the students to try different representations of data, others explore the benefits of random access to the code lines or changing the data used by the program. Our research agenda includes evaluating the value of explanations in the explanatory visualizations and the student attitude to our different attempts to make learning more active.

A simulation of Buffer Overflow

Control panel

Step Forward | Step Back | Reset

Click on any line of code to step directly to that line

Code Window

```
int main()
{
  int i;
  char line[10];
  int j;
  j =i=10;
  strcpy(line, "howdy");
  printf("i=%d, j=%d, line = %s\n", i, j, line);
  strcpy (line, "advertisement");
  printf("i=%d, j=%d, line = %s\n", i, j, line);
  strcpy (line, "imagination");
  printf("i=%d, j=%d, line = %s\n", i, j, line);
  i=1207960129;
  printf("i=%d, j=%d, line = %s\n", i, j, line);
  strcpy (line, "pretty thing");
  printf("i=%d, j=%d, line = %s\n", i, j, line);
  i=10;
```

Console Window

```
i=10, j=10, line=howdy
i=1701737472, j=10, line=advertisement
```

Stack/Memory Window

Address	Memory				
640	01101110 'n'	00000000 '\0'	01110100 't'	00000000 '\0'	int i
636	01100001 'a'	01110100 't'	01101001 'i'	01101111 'o'	char line[]
632	01100001 'a'	01100111 'g'	01101001 'i'	01101110 'n'	
628			01101001 'i'	01101101 'm'	
624	00000000	00000000	00000000	00001010	int j

Status/Explanation Window

Now the system is trying to copy the value "imagination" in to the array line. Note that the string is 1 byte longer than line[]. So the extra byte"n" will be written to the memory area of the variable i starting at 640. The null terminator is written at 641. Also note that the contents of 642 and 643 are not changed and they still remain "t" and "\0".

Figure 5. An environment for explanatory visualization of the buffer overflow problems.

Pointer Arithmetic

✓Decimal | Binary | Hex | Text

Program Code: Back | For

```
int main() {
    char x[3] = {'a', 'b', 'c'};
    char* y = x;
    printf("Element 0 = %c \n", *y);
    printf("Element 1 = %c \n", *(y+1));
    printf("Element 2 = %c \n", *(y+=2));
    printf("Element 1 = %c \n", *(--y));
    printf("Element 1 = %c \n", *(y++));
    printf("Element 2 = %c \n", *y);
    printf("Element ? = %c \n", *(y+1));
    return 0;
}
```

Program Output:

```
Element 2 = c
```

Address	+ 0	+ 1	+ 2	+ 3	Variable
60	097	098	099		x
56	000	000	000	62	y
52					
48					
44					
40					
36					
32					
28					
24					
20					
16					
12					
8	program	code	program	code	
4	program	code	program	code	
0	reserved	by the	operating	system	

Explanation:

It is also possible to use the += and -= operators increase or decrease the address and assign this new address to y. Here 2 is added to the address in y, 60, and the new address, 62, is assigned to y. This expression also returns the new address, which is subsequently dereferenced.

Figure 6. An environment for explanatory visualization of C Pointer problems

WADEIn: Some Classroom Study Results

We routinely run classroom studies of all educational tools developed in our lab. During the first semester of using a tool in the classroom we usually run a formative pilot study to uncover (through discussions with the students) obvious problems and to determine critical aspects to be evaluated. It helps us to design a questionnaire that we use to run a set of more formal studies in the following semesters. During the study we make a tool available (through the Web) for all students. Each tool is introduced during the lecture and often used as a teaching tool during a lecture. At the end of the course, we asked students to fill in a non-mandatory questionnaire. In our evaluation we only consider questionnaires filled by students who performed some sufficient amount of work with the system to understand and appreciate it. For example, in the case of WADEIn "sufficient work" means working with at least 20 expressions in both demonstration and evaluation mode. Since several versions of WADEIn have been used in the classroom over four semesters, we have results of three formal studies. Data from the most recent study provide some evidence in favor of our adaptive visualization is shown below. The data were collected from 29 students who tried between 20 to 441 different examples with WADEIn over the duration of the course. As shown in Figure 7, student attitudes to WADEIn and its features are very positive. More than 80% students found WADEIn and its adaptive nature helpful with about 30% students providing highly positive answers and none selecting negative answers. By our standards, this is a very impressive results making WADEIn one of the most useful tools we have ever developed.

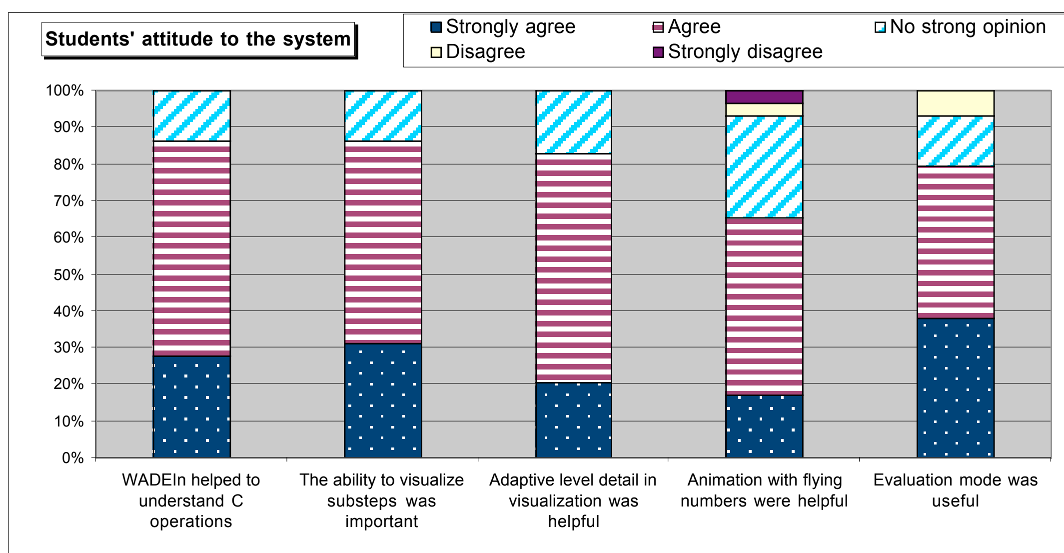

Figure 7. Student attitude to WADEIn and its features

Summary and Future Work

This paper discussed the problems of developing effective program visualization to be used in programming and data structure courses. We have explored engaging, adaptive, and explanatory visualization. In our current work we explore different ways to improve the effectiveness of visualization in different systems (and different modes inside WADEIn). The goal is to evaluate the separate "added value" of each in the most appropriate context. However, our true believe is that the best value can be provided by using these three promising ways at the same time. Indeed, they are not contradictory, but complementary. Explanatory visualization can be naturally used in several ways to engage students. For example, Kumar's problems always challenge the student with the task of predicting the answer before presenting the full explanation (Kumar, 2001; Shah & Kumar, 2002). Adaptation can empower both engaging and explanatory visualization. For example, to keep the student interest and motivation the system will challenge the student to predict the results of the least known execution steps as done in WADEIn's evaluation mode. Likewise, to prevent the student from drowning in the flow of explanations, the system can generate detailed explanations for the least known operations. In our future work we plan to explore different combinations of engaging, adaptive, and explanatory visualization. Our ultimate goal is to maximize the value of educational visualization for every student. In addition, we are interested in using engaging, adaptive, and

explanatory visualization in some contexts different from program visualization. We believe that the benefits of these powerful ways to expand visualization can be demonstrated in multiple domains.

References

M. H. Brown and M. A. Najork. "Collaborative Active Textbooks: A Web-Based Algorithm Animation System for an Electronic Classroom"; In *Proceedings of the IEEE Symposium on Visual Languages (VL'96)* (pp. 266-275). 1996.

P. Brusilovsky. "Adaptive visualization in an intelligent programming environment"; In *Proceedings of the East-West International Conference on Human-Computer Interaction* (pp. 46-50). ICSTI. 1992.

P. Brusilovsky. "Program visualization as a debugging tool for novices"; In *Proceedings of the INTERCHI'93 (Adjunct proceedings)* (pp. 29-30). 1993.

P. Brusilovsky. "Explanatory visualization in an educational programming environment: connecting examples with general knowledge"; In *Proceedings of the 4th International Conference on Human-Computer Interaction, EWHCI'94* (pp. 202-212). Berlin: Springer-Verlag. 1994.

P. Brusilovsky, E. Calabrese, J. Hvorecky, A. Kouchnirenko, and P. Miller. "Mini-languages: A way to learn programming principles"; *Education and Information Technologies*, vol. 2(1), pp. 65-83. 1997.

J. E. Butler and J. B. Brockman. "A Web-based learning tool that simulates a simple computer architecture"; *SIGCSE Bulletin - inroads*, vol. 33(2), pp. 47-50. 2001.

M. D. Byrne, R. Catarambone, and J. T. Stasko. "Evaluating animations as student aids in learning computer algorithms"; *Computers & Education*, vol. 33(5), pp. 253-278. 1999.

J. Domingue and P. Mulholland. "An Effective Web Based Software Visualization Learning Environment"; *Journal of Visual Languages and Computing*, vol. 9(5), pp. 485-508. 1998.

J. Haajanen, M. Pesonius, E. Sutinen, J. Tarhio, T. Teräsvirta, and P. Vanninen. "Animation of user algorithms on the Web"; In *Proceedings of the VL '97, IEEE Symposium on Visual Languages* (pp. 360-367). IEEE. 1997.

T. Hampel, R. Keil-Slawik, and F. Ferber. "Explorations - A New Form of Highly Interactive Learning Materials"; In *Proceedings of the WebNet'99, World Conference of the WWW and Internet* (pp. 463-468). AACE. 1999.

C. D. Hundhausen, S. A. Douglas, and J. T. Stasko. "A meta-study of algorithm visualization effectiveness"; *Journal of Visual Languages and Computing*, vol. 13(3), pp. 259-290. 2002.

A. N. Kumar. "Learning the interaction between pointers and scope in C++"; In *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'2002)* (pp. 45-48). ACM Press. 2001.

A. McKenna and A. Agogino. "Engineering for middle schools: A Web-based module for learning and designing with simple machines"; In *Proceedings of the FIE'97, Frontiers in Education Conference* (pp. 1496-1501). Stipes Publishing L.L.C. 1997.

H. Shah and A. N. Kumar. "A tutoring system for parameter passing in programming languages"; ACM SIGCSE Bulletin, 34, 3 (2002), 170 - 174.

J. Stasko, A. Badre, and C. Lewis. "Do Algorithm Animations Assist Learning? An Empirical Study and Analysis"; In *Proceedings of the INTERCHI'93* (pp. 61-66). ACM. 1993.

S.-H. S. Tung. "Visualizing Evaluation in Scheme"; *Lisp and Symbolic Computation*, vol. 10(3), pp. 201-222. 1998.

D. Yaron, R. Freeland, D. Lange, and D. J. Milton. Using Simulations to Transform the Nature of Chemistry Homework. Carnegie Mellon University, Pittsburgh (2001),