WADEIn II: A Case for Adaptive Explanatory Visualization

Peter Brusilovsky and Tomasz D. Loboda
School of Information Sciences, University of Pittsburgh
Pittsburgh, PA 15260
{peterb, tol7} @ pitt.edu

ABSTRACT
Adaptive explanatory visualization is an attempt to integrate two promising approaches to program visualization: adaptive visualization and explanatory visualization. The goal of this paper is to demonstrate the ideas of adaptive explanatory visualization using a practical example. The paper introduces the WADEIn II system for the visualization of expression evaluation in the C programming language, shows how expression evaluation visualizations can be made adaptive, and explains our approach to the adaptive generation of explanations.

Categories and Subject Descriptors
I.6.8 [**Simulation and Modeling**]: Types of Simulation – *animation, combined, visual*; K.3.2 [**Computers and Education**]: Computer and Information Science Education – *computer science education;*

General Terms
Design, Human Factors, Languages

Keywords
Adaptive visualization, program visualization, explanations, user modeling, expression evaluation

1. INTRODUCTION
Program visualization is considered to be one of the most powerful educational tools in Computer and Information Science education. Visualization can provide a clear visual metaphor for explaining complicated concepts and uncovering the dynamics of important processes that are usually hidden from the student's eye. For many years, the focus of visualization research has been upon developing better tools and exploring new contexts. Relatively few studies of the effectiveness of visualization have been performed because the benefits of visualization seemed obvious to any researcher or teacher. Yet, several experiments [4; 12] have shown just the opposite; that the educational benefits from observing visualizations are unexpectedly low. Often, the presence of a well-developed visualization failed to help students understand what was taking place inside a program or an algorithm. Since discovering this, several researchers and teams have focused their efforts on the constructive use of visualization. We can distinguish three promising approaches to "useful visualization," which are: *engaging*, *explanatory*, and *adaptive* visualization.

ITiSCE'06, June 26–28, 2006, Bologna, Italy.

The idea of *engaging visualization* [10] is to change the students from passive observers to active learners by engaging them in some activity related to visualization. Several options have been explored, ranging from "light activities," such as having students develop their own data sets for a given visualization to "heavy activities," such as asking the student to construct the whole visualization themselves, instead of watching a "prepared" one. Most of these innovations brought positive results [4; 6; 7].

The idea of *explanatory visualization* is to augment every animated step within the visualization with natural language explanations. The role and content of these explanations center on describing what is going on, why it happens, and how it relates to general programming principles. The need to supplement visualization with explanations was first expressed over 10 years ago [1; 2; 12], and early studies demonstrated that explanations do indeed help the students understand what they see [2]. More recently, Kumar [9] suggested a model-based framework for the dynamic generation of explanations, exploring it in a set of *problets* [5; 8; 11]. In multiple evaluations, the generated explanations have been shown to be effective for improving student learning.

Adaptive visualization is based on the assumption that a student may have unequal levels of knowledge about the different *elements* of a program or algorithm that is being visualized. The theory behind adaptive visualization is to inversely match the level of detail in the visualization of each construct or step to the student's level of knowledge about it: The lower the student's level of understanding about a construct, the greater the level of detail in its visualization. This approach allows a student to focus attention on the least understood components while still being able to understand the whole visualization. Our early studies with a simple mini-language confirmed that adaptive visualization improves the student understanding of the visualization [1]. More recently, we developed the WADEIn system [3] to explore the value of system-adapted visualization in the context of the C-programming language. Several classroom studies of WADEIn brought encouraging results: more than of the 80% students in our classes found the WADEIn system and its adaptive visualizations *helpful* or *very helpful*.

The focus of our current work is the integration of two beneficial visualization approaches – adaptive and explanatory visualization. In order to explore this combination, we went on to develop the

WADEIn II system. In WADEIn II we redesigned the adaptive visualization, basing it on the previous version of the system while extending it with explanations that are generated using a variant of Kumar's model-based approach. It is important to realize that WADEIn II does not simply add together adaptation and explanation, it attempts to integrate them. The very content of the generated explanations is adapted to the changing level of the user's knowledge. WADEIn II was implemented and pilot-tested in the Fall of 2005 and is currently being used in an introductory programming course.

The goal of this paper is to demonstrate adaptive explanatory visualization using the expression evaluation context supported by WADEIn II. Expression evaluation is a relatively complicated topic that has rarely been supported by visualization tools. In this paper, we present WADEIn II, show how expression evaluation visualization can be made adaptive, and explain our approach to the adaptive generation of explanations.

2. THE OBJECT OF VISUALIZATION

WADEIn II is a Web-based visualization tool for students learning the C language in introductory programming courses. It uses adaptive visualization and textual explanations to portray the process of expression evaluation. It supports twenty-four C operators, e.g. addition, modulo, and less-or-equal (although WADEIn II does not address pointer arithmetic problems). The system visualizes and explains these operators (which are explicit concepts) as well as several implicit concepts that may appear in that context. The progress that students make with explicit concepts is being tracked and is shown to them.

Concepts modeled implicitly are: (1) reading a variable, (2) implicit casting, and (3) logical value representation. For example, the *logical value representation* models the fact that in the C language, zero is treated as false and any non-zero value is treated as true. This concept may appear in the context of many operators, e.g. *less-than*, *equal*, or *logical OR*. The student's knowledge of implicit concepts is traced, but not shown to them, because implicit concepts are more difficult to identify.

3. STUDENT'S VIEW

3.1 User Interface Segmentation

The user interface (*Figure 1*) is divided into four regions: Goals and Progress (*A*), Settings (*B-E*), Navigation (*F-H*), and the Blackboard (*I-M*). The *Goals and Progress* region contains a list of explicit concepts being learned, along with progress indicators known as *skillometers* that allow students to monitor their own progress.

The *Settings* region appears even before expression evaluation is begun. First of all, it allows the student to pick an expression to be evaluated. They can either select a predefined expression or enter their own expression. They can also change the initial values of the variables that appear in the expression, e.g. expression *B % 7* contains variable *B*. Finally, the student can switch between system modes, that is, *exploration* and *knowledge evaluation*. In

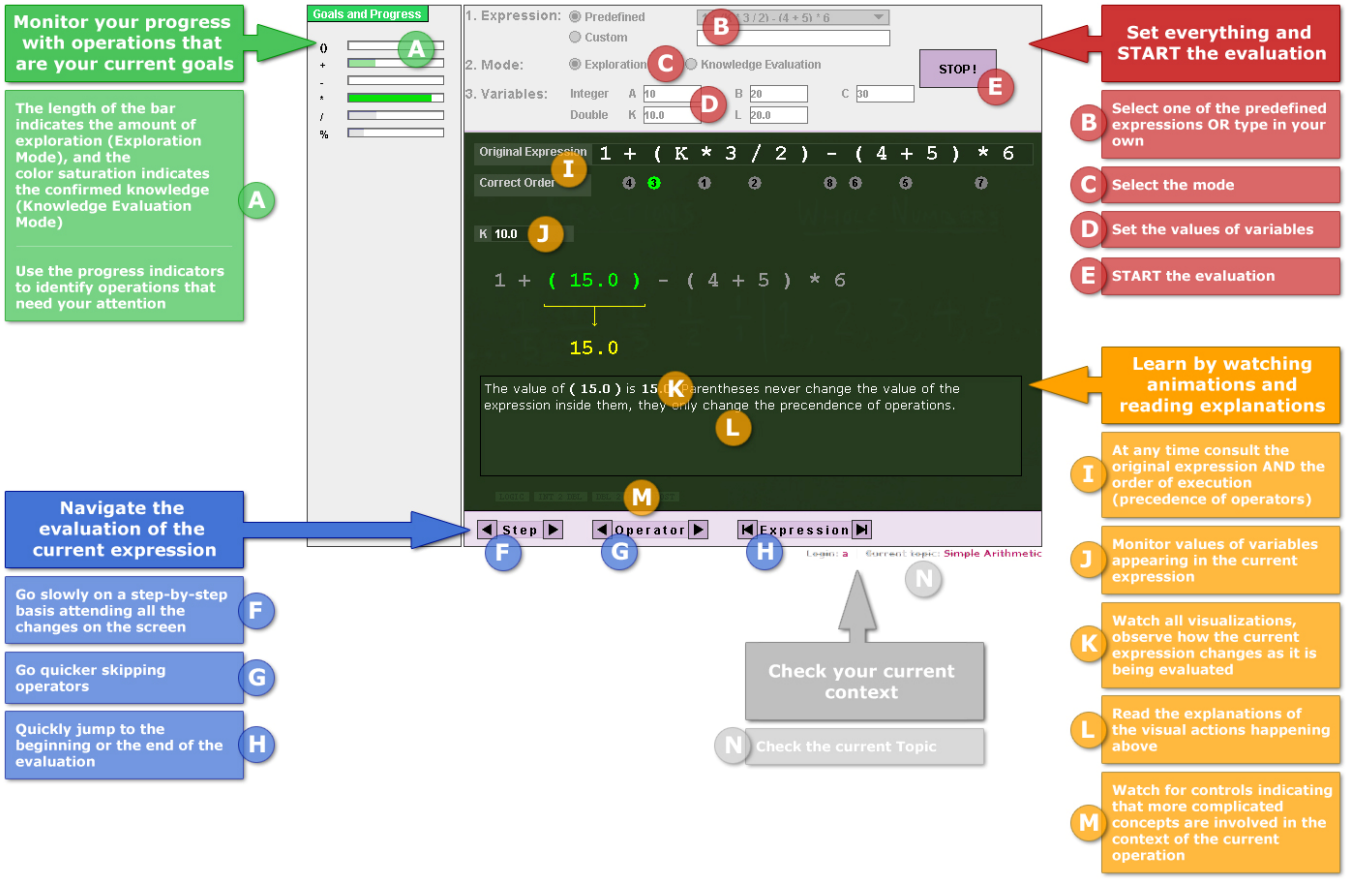

Figure 1. The user interface of WADEIn II system – exploration mode.

the exploration mode, the student is presented with the order of evaluation (as restricted by the precedence of operators) and all the visualizations and explanations the system has to offer. It is the exploration mode that the student will start to use when learning a new concept. In this mode they will be building the corpus of their knowledge. The second mode, the knowledge evaluation mode, has been designed to let students check their level of knowledge. First, the student has to indicate the order of evaluation they believe to be correct. Then they are shown the correct order and can compare it with their own answer. Next, students are asked to provide values for all operations in the current expression. These two modes serve different purposes. In the exploration mode, knowledge is accumulated and organized. In the knowledge evaluation mode it is checked. This separation is also maintained on the student modeling level, which is described in Section 3.4.

The *Navigation* region lets the student navigate the expression evaluation. It can be done on a step-by-step or operator-by-operator basis, whether forward or backward. Additionally, it is possible to quickly jump to the beginning or the end of the evaluation.

The *Blackboard* region is where all visualizations and explanations are presented. Both are explained in detail in Sections 3.2 and 3.3. Additionally, this region displays the original expression, the order of evaluation, and values for the variables in the expression currently being evaluated.

3.2 Visualizations

WADEIn II is capable of showing the evaluation of every operation in detail. Depending on the kind and complexity of the operation, its visualization can include up to five stages, as introduced below. Each stage may include several steps. Many steps are animated to help students follow changes. The system adapts the speed of those animations to the progress of the student. More adaptation details are given in section 3.4. Different colors are used to contextualize and group visual events. Green is used to highlight the current operation. Yellow is used to visualize the value of that operation. Red is used to visualize the reading and writing of variable values and finally, cyan is used in pre- and post-visualizations. Each color carries information about its specific context.

1. **Variable read:**
   If the current operation contains variables, their values need to be read before the evaluation can continue. Reading a variable is visualized in three steps. First, the name of the variable is highlighted in red. Second, an animation shows that the current value of that variable is being inserted into the operation. That value will also be shown in red. Third, after the value has been inserted, its color changes from red to green.
2. **Value production:**
   The evaluation of many operations yields a number. This number replaces the operation. For instance, *2 && 0* yields *0*, therefore *0* will replace *2 && 0* in the expression. This process is visualized in four steps. First, the operation (*2 && 0*) is highlighted in green. Second, its value (*0*) is shown in yellow. Third, the value replaces the operation in an animation. Fourth, inserting a value into an expression changes the color from yellow to green.
3. **Variable write:**
   The evaluation of assignment operations yields a number. For example, *A = 7* yields *7*. However, apart from visualizing the value (see *VALUE* described above) a side effect, the change in the variable value, needs to be visualized as well. This is done in three steps after the value (*7*) of the operation has been shown (the first step of VALUE visualization). First, the value (*7*) is highlighted in red. Second, an animation shows that this value becomes the new value of the variable in question (variable *A* will receive the value *7*). Third, the value highlighted in red changes back to yellow. At this point the *VALUE* visualization can be resumed.
4. **Pre:**
   Pre-increment and pre-decrement operations change the value of a variable. For example, the *++A* operation will add *1* to the current value of variable *A*. After the value has been changed, the new value has to be read into the expression. This process is visualized in four steps. First, the operation (*++A*) is highlighted in cyan. Second, an animation shows that this operation changes the value of the variable in question (*A*). Third, another animation shows that the new value of the variable is read into the expression. That value will also be shown in cyan. Fourth, inserting the value changes the color to green.
5. **Post:**
   Post-increment and post-decrement operations change the value of a variable. For example, the *A--* operation subtracts *1* from the value of variable *A*. That subtraction happens after the rest of the evaluations in the current expression have ended. This is visualized in six steps. First, the operation (*A--*) is highlighted in cyan. Second, an animation notes that the value of the variable in question (*A*) will be changed later. Third, another animation shows that the current value of the variable (*A*) is read into the expression. That value will also be shown in cyan. Fourth, inserting the value changes the color to green. Fifth, after the rest of the evaluation ends, the variable in question (*A*) is highlighted in cyan. Sixth, an animation shows that the previously remembered operation (subtract 1) is finally executed, changing the value of the variable (*A*).

Some of the concepts in our domain are more difficult than others. We thought it would be beneficial to make the student aware that they are dealing with difficult concepts. We identified the following four concepts as difficult: (1) logical values representation, (2) integer to double implicit cast, (3) double to integer implicit cast, and (4) post-decrement and post-increment.

Our system features four visual flags for the four most difficult concepts. The flags are highlighted in orange when a particular concept is encountered. For instance, the *logical values representation* flag will be highlighted in the context of the *less-than* operator, to remind the student that in the C language, zero denotes false and any non-zero value denotes true. Please note that the flags cannot be used by the student to monitor their

progress. They do not represent the student's knowledge level. They are only visual indicators or reminders of the difficulty of the subject and are used only in the context of the current operation.

3.3 Explanations

There is no guarantee that a particular set of visualizations will be easily understood by all students. Some visualizations can be confusing, especially when the students do not know the material. The addition of a natural language description of these visual events can be a very helpful way of providing the necessary clarification.

WADEIn II is equipped with textual explanations that accompany visualizations. Each visualization step has an explanation. Some of the explanations may be empty. For example, the *2 + 3* operation is visualized in four steps (as in the *Value* visualization from the previous section): highlight operation, show value, animate value, and insert value. When the operation is highlighted, a short introduction to the summation operator is presented. Next, when the value is shown, an explanation is given that *2* and *3* sum up to *5*. The explanation does not change in the next step when the value is being animated, to limit the number of changes on the screen to one. Otherwise, students may get confused as to which change they should be attending. This is especially important when an animation is taking place. A visually subtle change, like changing explanatory text, may not be noticed while movement is taking place elsewhere on the screen simultaneously. The explanation for the last step of *2 + 3* operation is empty, because the summation operator was considered easy enough to not require any additional explanations at this step of the visualization. More difficult operators, such as modulo, would have had additional text presented.

Each explanation is constructed from one or more fragments of text. Each fragment addresses a different idea. The system decides which fragments to present depending upon the student's level of knowledge. This adaptive behavior is described in detail in section 3.4. The quality of explanations is very important. Bad wording can increase a student's confusion instead of helping them to understand the material better. The explanations used in WADEIn II have been developed by a group of programming and education experts.

3.4 Adaptation

Informational needs change as the learner gains more insight into the learning material. For example, the basics are very important at the beginning, but gradually become less and less relevant. At a certain point they start becoming irrelevant – the student has mastered them and would not benefit from seeing them any more. Overly detailed presentations might discourage the student. They might stop using the tool convinced that there is nothing more they could learn. They also might get frustrated by being forced to see things that they already know, over and over again. That is why keeping track of the student's progress and adapting to it is a very important feature of an educational tool.

Each concept in WADEIn II is described in terms of *exploration knowledge* (k_{ex}) and *evaluation knowledge* (k_{ev}). These types of knowledge represent the student's progress in the two modes available to the system. Progress for easier concepts should happen quicker than for the more difficult ones. That is why each concept is additionally described by a complexity level, which is defined by the domain expert.

WADEIn II uses five types of student activity as input to student models:

- ***oex***: The student sees the **o**rder of evaluation (**ex**ploration mode).
- ***ex***: The student sees the visualization of a particular operation (**ex**ploration mode).
- ***oev***: The student indicates the **o**rder of evaluation (knowledge **ev**aluation mode).
- ***evOk***: The student provides a correct value of an operation (knowledge **ev**aluation mode).
- ***evErr***: The student provides an incorrect value of an operation (knowledge **ev**aluation mode).

By default, the system assumes the student has no initial knowledge (k_{init}) of any concept. However, the student can set his/her initial level when registering for the system.

The formulas for the two types of knowledge are:

$$k_{ex,i} = k_{init,i} + \frac{g_{ex} * n_{ex,i} + g_{oex} * n_{oex,i} - l_{ex} * n_{evErr,i}}{\sqrt{c_i}},$$

$$k_{ev,i} = k_{init,i} + \frac{g_{ev} * n_{evOk,i} + g_{oev} * n_{oev,i} - l_{ev} * n_{evErr,i}}{\sqrt{c_i}},$$

where i is the index of the concept, g is the knowledge gain, l is the knowledge loss, n is the number of times a particular student activity has occurred, and c is the complexity of the concept. Levels of both types of knowledge can range from 0 to 5. The g and l are parameters of the model and define its sensitivity. Higher values of g and lower values of l will cause the model to reach the upper bound (5) more quickly.

The two knowledge levels are represented differently on the skill-o-meters: Exploration knowledge is indicated by the length of the progress bar while evaluation knowledge is indicated by the intensity of the bar's color (the higher the level of knowledge the more intense the color). That distinction helps students plan their goals better. For instance, they can see that even though they have made a considerable amount of progress in the exploration mode they have not yet confirmed their knowledge of that concept in the evaluation mode.

As the student progresses, the behavior of the system changes. The speed of the visualization animations gradually increases until finally the animations are replaced by a single-step action. Explanations are constructed from fragments of text. Each fragment is relevant only until the student reaches a certain level of knowledge. After that threshold has been reached, that particular fragment is hidden and is no longer a part of that explanation. At a certain point, no explanations are shown any more.

Note that the above formulas assume the linear change (gain or loss) of knowledge. This is a simplification that was reused (with some updates) from the earlier, successful version of the system

[3]. It is one of the future goals of our project to identify the most appropriate student modeling approach for adaptive visualization.

4. INSTRUCTOR'S VIEW

To enable instructors to tune the system to their preferred way of teaching expressions, WADEIn II allows an instructor to divide the set of all the C language operators into *subsets* (called topics) and introduce them to students topic by topic. For each topic, the instructor can create a set of exercises. Each exercise includes a set of suggested expressions and specifies the system mode: *exploration* or *knowledge evaluation*. Using that approach, it is possible to create two separate sets of expressions (exploration and evaluation of concepts) in the same topic. Because of that, the teacher is able to evaluate student knowledge on different expressions than were used in the exploration mode.

5. IMPLEMENTATION

WADEIn II is a client-server Web application deployed in a classic three-tier architecture. The front-end is a Java applet and the back-end, a Java servlet connected to a relational database. The servlet initializes the applet and later serves as an intermediate layer bridging the user interface and the database. The database stores all the system settings, textual explanations and user models. Java 1.4.2 has been used and the system may be accessed from any machine equipped with an Java-enabled web browser, without the need to install any additional software. The system is accessible at:

http://www.sis.pitt.edu/~paws/system_wadein.htm.

6. CONCLUSIONS AND FUTURE WORK

WADEIn II demonstrates how visualization can be made adaptive and how adaptive explanation can be added to it, in the specific case of expression evaluation. For our team, this work is a component of a large-scale project on studying adaptive explanatory visualization. In the context of this project, we want to explore this innovative kind of visualization, in order to introduce different concepts in several contexts. Through multiple studies, we want to determine how to make visualization more useful through adaptation and generated explanations. In our project, we continue to collaborate with the team of Dr. Kumar at Ramapo College of New Jersey, and we welcome other teams to join our efforts. We hope that this paper has provided sufficient details to engage researchers and practitioners who may be interested in adaptive explanatory visualization, motivating further research on this topic.

7. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 0426021. We would like to thank Sergey Sosnovsky, Rosta Farzan, and Roman Bednarik for their valuable input. Additionally, we would like to thank anonymous reviewers.

8. REFERENCES

[1] Brusilovsky, P. Program visualization as a debugging tool for novices. In: Proc. of INTERCHI'93 (Adjunct proceedings), (Amsterdam, 24-29 April 1993), 29-30.

[2] Brusilovsky, P. Explanatory visualization in an educational programming environment: connecting examples with general knowledge. In: Blumenthal, B., Gornostaev, J. and Unger, C. (eds.) Human-Computer Interaction. Lecture Notes in Computer Science, Vol. 876. Springer-Verlag, Berlin, 1994, 202-212.

[3] Brusilovsky, P. and Su, H.-D. Adaptive Visualization Component of a Distributed Web-based Adaptive Educational System. In: Intelligent Tutoring Systems. Vol. 2363. Springer-Verlag, Berlin, 2002, 229-238.

[4] Byrne, M.D., Catarambone, R., and Stasko, J.T. Evaluating animations as student aids in learning computer algorithms. Computers & Education, 33, 5 (1999), 253-278.

[5] Dancik, G. and Kumar, A.N. A tutor for counter-controlled loop concepts and its evaluation. In: Proc. of 2003 Frontiers in Education Conference (FIE 2003), (Boulder, CO, November 5-8, 2003), Session T3C.

[6] Hundhausen, C.D., Douglas, S.A., and Stasko, J.T. A meta-study of algorithm visualization effectiveness. Journal of Visual Languages and Computing, 13, 3 (2002), 259-290, available online at http://lilt.ics.hawaii.edu/~hundhaus/writings/VL2000-Experiment.pdf.

[7] Jarc, D.J., Feldman, M.B., and Heller, R.S. Assessing the benefits of interactive prediction using Web-based algorithm animation courseware. ACM SIGCSE bulletin. 32, 1 (2000), 377-381.

[8] Kumar, A.N. Learning the interaction between pointers and scope in C++. In: Proc. of 6th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'2002), (Canterbury, UK, June 25-27, 2001), ACM Press, 45-48.

[9] Kumar, A.N. Model-based generation of demand feedback in a programming tutor. In: Kay, J. (ed.) Supplementary Proceedings of the 11th International Conference on Artificial Intelligence in Education (AI-ED 2003). IOS Press, Amsterdam, 2003, 425-432.

[10] Naps, T.L., Eagan, J.R., and Norton, L.L. JHAVE – an environment to actively engage students in Web-based algorithm visualizations. ACM SIGCSE bulletin. 32, 1 (2000), 109-113.

[11] Shah, H. and Kumar, A.N. A tutoring system for parameter passing in programming languages. ACM SIGCSE bulletin. 34, 3 (2002), 170-174.

[12] Stasko, J., Badre, A., and Lewis, C. Do Algorithm Animations Assist Learning? An Empirical Study and Analysis. In: Proc. of INTERCHI'93, (New York, Amsterdam, 24-29 April 1993), ACM, 61-66.