Survey of Effortlessness in Algorithm Visualization Systems

Ville Karavirta, Ari Korhonen, and Petri Tenhunen
Helsinki University of Technology
Department of Computer Science and Engineering
Finland
{vkaravir, archie, ptenhune}@cs.hut.fi

Abstract

This paper reports the results of an on-line survey conducted among computer science educators to examine effortless creation of algorithm visualizations. Based on the results, we give a proposal for measuring effortlessness in this sense. The aim is to enhance the understanding of the visualization tools adequate in computer science education.

1 Introduction

The idea of using visualization to promote the understanding of abstract concepts, like data structures and algorithms, has become widely accepted. Despite its benefits, algorithm visualization (AV) has failed to become popular in mainstream computer science (Stasko, 1997; Baecker, 1998). One of the main obstacles for fully taking advantage of AV systems seems to be the time and effort required to design, integrate and maintain the visualizations (Naps et al., 2003b). According to Hundhausen et al. (2002) the process of creating AVs is thought to be too laborious to be worthwhile. Thus, "a future challenge is to create tools and methodologies which will result in the use of SVs by the majority of computer science educators" (Domingue, 2002).

Several attempts have been made to introduce a system that levels out the burden of creating new visualizations (e.g. Haajanen et al., 1997; Korhonen and Malmi, 2002; LaFollette et al., 2000; Naharro-Berrocal et al., 2002). However, none of these systems has gained wide recognition. In our previous study (Karavirta et al., 2002), we examined four systems, in order to identify why the creation of software visualization is such a laborious process. We wanted to emphasize the instructor perspective because "the visualization research has focused on the developer and designer while research in CS education has focused on [...] student learning. In contrast, virtually no research has focused on the needs of the instructor" (Naps et al., 2003b). However, it is the instructor that plays the key role in taking the AV system in use.

Unfortunately, effortlessness is a highly subjective measure including many factors. For example, the different systems can be put on the same line only by first determining the context where the systems are utilized. With one system, it might be easy to create static visualization from an already existing code whereas another system might be effortless in the sense that the animations can be created on-the-fly in a lecture situation. Creating visualizations with an inadequate system for a certain task might require a lot of extra effort compared with more feasible system. Thus, algorithm visualization systems are often seen laborious because people are drilling with a hammer. More research is needed to identify the essence of effortless creation of algorithm visualizations.

In this study, we have created a questionnaire for computer science educators to sort out the elements of effortless creation of algorithm visualization. In our survey, we concentrate on the instructor's point of view to identify the typical use cases when visualizations are created, and typical expectations when starting to use an algorithm visualization system. In the rest of the paper, we describe the survey, results, give a proposal for how to measure effortlessness in AV systems, and finally make some conclusions.

2 Survey

The starting point for this research was the work done by the working group on "Exploring the Role of Visualization and Engagement in CS Education" at the ITiCSE 2002 conference (Naps

et al., 2003a). The results from the three surveys evaluated in that paper (Grissom's survey from ITiCSE'00, ITiCSE'02 WG pre-conference on-line survey, and ITiCSE'02 WG Index Card survey) motivated us to develop a new, more detailed on-line survey from the *effortlessness* point of view. Using some items from the three surveys, we refined a survey of our own that was first tested with the attendees of the Third Program Visualization Workshop (PVW'04). The preliminary results were also presented there. However, our survey continued after the conference and we advertised the research in various mailing lists (*e.g.*, for PPIG (Psychology of Programming Interest Group), SIGCSE (ACM Special Interest Group on Computer Science Education), and PVW members) to get a more versatile sample. The final results are reported in this paper. Respondents were from USA (7), UK (5), Finland (2), Germany (2), Israel (2), New Zealand (2), Island (1), and Spain (1), which makes total of 22 responses. This is less than in the previous syrveys. Moreover, as the terminology in the field varies it causes some problems to collect reliable data. For example, some of the respondents did not answer the survey as they felt that the survey covered only very narrow set of courses.

In the ITiCSE WG survey they asked much more detailed questions on the background of the respondent as well as their attitude towards AV. We expected to reach a similar population, thus we concentrated more on how they have applied AV in practice. The only background questions, in addition to the above, were the ones on subjects taught and in which course levels. The subjects varied from basic programming courses to computer graphics, and from computer architecture to discrete mathematics. Moreover, all course levels were almost evenly distributed as many of the respondents were teaching several courses. After the few background questions, previous usage of AV systems and usage in teaching were asked.

Five respondents denied the use of algorithm visualizations in their teaching, three of whom have not used any tools to create algorithm visualizations. However, there is a total of six respondents that have not used AV tools to create algorithm visualization, thus three respondents must have used them, but not in teaching.

The respondents who have used AV system(s) were asked about the origins of the system(s) they have used. We have 25 responses from 18 respondents. Ten have developed a tool or system by themselves. One of these, however, has never used it to create algorithm visualizations (but possibly used it in teaching, see the previous questions). Six have developed system(s) in a team within a single institution. Four respondents have developed a new tool by reusing existing tools or libraries. Again, four of the respondents uses software developed by researchers in other institutions. Only one respondent used software developed by some other team within his or her own institution. None of the respondents selected "Commercial software" even though some examples of these are mentioned later in free text responses.

The rest of the survey was divided to two phases. First, we wanted to collect free text descriptions on the actual use cases and situations in which the respondents were using AV. After the first phase, the respondent was directed to another page where we provided ready-made scenarios and asked the respondents to state how well these apply in their context. The whole survey, with some additional data not presented in this survey, can be located on the web at `http://www.cs.hut.fi/Research/SVG/survey/`.

3 Results

Most respondents described several use cases where they, as teachers, could create and use AV – 16 respondents and 28 use cases. Even though in most cases the task was clearly identifiable, the ultimate goal within the use case was often blurred. Some typical tasks, however, can be identified from the data (i.e., *tracing and debugging*, *problem solving and exercises*, *data abstraction and representation issues*, and *teaching data structures and algorithms in general*).

By lowering the abstraction level, we can also identify the context in which AV is used. The following items were mentioned with more than one use case (number of use cases where the context occurred is marked in parenthesis): Programming – data structures, data flow, and

control flow (6); Sorting algorithms (4); Graph algorithms (4); Mathematics / theoretical CS (3); and Geometric algorithms (2). Thus, tracing and debugging of an implemented algorithm (i.e., real execution) is the most commonly mentioned use case. Albeit important one, the *actual implementation* of an algorithm, however, was not an issue in most of the use cases.

The tools that respondents have used could be divided into three groups: commercial products, visualization and graphics packages, and non-commercial AV systems. Commercial systems mentioned in the responses were PowerPoint, Visual Basic, and QuickTime. Visualization packages were the Visualization ToolKit (VTK), GraphViz, and OpenGL. Non-commercial products, however, was the most common category with products like Jeliot, JAWAA, Animal, JHave, TRAKLA2, MatrixPro, Poplog, SimAgent, JFLAP, and several Java applets.

According to respondents, the most common pros of the visualization systems were the features allowing to create animations easily, and sufficient navigation possibilities in the final animation. The lack of these very same properties were mentioned as cons of some other systems. Not so commonly mentioned pros were: easy installation, freely available, modular structure, and the fact that use of these tools might activate thinking. Respective cons were: limited domains where the system can be used and the observation that the interaction with such tools may be mechanical and therefore decrease thinking. A benefit mentioned especially with programming tools was that "since it's a programming language, I can get it to do just about anything I want".

The last question in the first part of the survey asked the respondent to describe an ideal tool that could be used to complete the use case. The responses were more of a wish-list about the system features or descriptions of current systems than any new visions about possible future tools. The most-often mentioned features are represented in Table 1.

Table 1: The ideal tools.

Category name	**Examples**	**Count**
Ease-of-use	Easy generation, own input data, customization of visualizations, example generators	7
Programming	Java, JavaScript, drag&drop coding	5
Special features	For beginners and experts, flexible notation and timing, interaction, visual effects	5
Abstraction level	General tools, small tools combined, many representations	4
Professionalism	Help functionality, internationalization, better user interfaces	3
Formats	*e.g.*, Flash, common file formats between systems	2

The respondents were also asked how much time they can afford to use and how much time they have used for the whole course and for different teaching related tasks. The tasks were developing the course contents (lectures, exercises, demos, text, ...), and several AV related tasks (*i.e.* searching, installing, and learning to use for good AV examples and tools, developing visualizations, adapting visualizations to the teaching approach and/or course content, solving problems in using the tool, and setting up visualizations in the classroom). The number of responses for this question was 19.

In most cases there was no difference between the time used and time available. However, when looking at the responses on the item "searching for AV tools", a clear difference was found. People have more time to search AV systems, than they are actually using for that.

The respondents were asked how much time they have available for predefined use cases and how much it would actually take to complete those use cases. The tasks were divided into four categories based on the situation: lecture, practice session, producing teaching material and examination/summative evaluation. The categories and the corresponding use cases are shown in Table 2. The number of respondents who answered this question was 20.

For each use case the respondents were asked how relevant it is according to the course he or she is teaching. The scale was from 1 (not relevant at all) to 5 (very relevant). The number of respondents that thought a use case is relevant (answers 3-5) is shown in Table 2

(column Rel). In addition, there is the column (Most Rel) that represents the number of respondents that consider the use case to be within the three most relevant tasks in their teaching. Moreover, the respondent was asked to state whether or not AV could be used to reduce the required work to complete the use case (column AV). The results show that AV is considered most useful to produce lecture examples and on-line illustrations. These two tasks are also considered to be the most relevant ones.

Table 2: Predefined use cases related to teaching. The columns Rel, Most rel, and AV show the numbers of responses in which the use case is considered to be relevant to the respondent, within one of the three most relevant use cases, and considered to reduce the work load of the use case, respectively.

Use case	**Rel**	**Most Rel**	**AV**
Lectures			
single lecture example	14	6	9
answering students' questions during a lecture	14	2	5
preparing questions/problems for a lecture	14	1	4
Producing teaching material			
on-line illustrations (static or dynamic)	12	5	8
text book/lecturer's notes illustrations (static)	12	1	3
Examination/summative evaluation			
creating exercises for examination (that students solve)	12	1	3
Practice session			
creating exercises for practice session (that students solve)	12	2	4
preparing a demo for a closed lab such that a tutor shows	9	3	6
preparing a demo students interact with in closed labs	7	3	5
preparing a demo students interact with in open labs	6	4	3

The section included also a question about the best tool the respondent has used for the use case, and the three most important features of the system. The features mentioned more than once were "saves time", "must be reliable", "one has control over the final visualization" (for example, possibility to move backwards and forwards, pause, play the animation), "allows student input data", "helps explaining/understanding", and "easy/automatic creation". There was also one response that stated that "all [systems] have increased my workload".

Most of the use cases are estimated to be completable just within the time available. Creation of text book illustrations and demos for closed labs was estimated to take a litle more time than what is available. A clear difference was observed in on-line illustrations. The average time to complete such a use case was 0.69 hours, whereas the time available was 1.23 hours.

The next question was about the content generation approaches. Table 3 lists the different approaches and illustrates that the approaches were almost equally preferred.

Table 3: Preferred content generation approaches.

Content generation approach	**Count**
Interactive simulation (user can experiment with methods; easy for implemented concepts, new concepts require understanding of underlying API)	11
Fully prepared examples (easy to use, no work; not adaptable)	9
"Generators" for some AV system (need only specify parameters, easy to use; restricted to certain topics, limited adaptability)	9
Automatic generation based on source code (easy to use, no work; requires source in appropriate language, display usually not adaptable)	7
Generation using a script (adaptable and designable; extra work)	7
Visual generation by drag and drop (highly adaptable, extremely flexible; takes much time, changing a value may not adapt the algorithm)	7

The last question of the survey was what are the tasks where a Graphical User Interface (GUI) is essential and when it is not needed at all. Tasks in which a GUI is needed according to the respondents are code/algorithm simulation, and customizing the visualization. Tasks that GUI is not essential in are automatic production from code, and textual tracing (for example, visualizing procedure calls).

4 Proposal for Measuring Effortlessness

In our previous study (Karavirta et al., 2002), we defined the effortless creation of algorithm visualization based on the following two criteria: the possibility to use the system on-the-fly basis, and available WYSIWYG user-interface. In addition, some of the categories presented in Taxonomy of Price et al. (1993) were applied. Moreover, we compared several systems with each other in order to discuss the relevance of such criteria. However, as we have already stated in our previous paper, our original criteria set are quite subjective in their nature. One of the main obstacles here is that the systems evaluated are usually targeted to different use cases, and the comparison among them is unreasonable. Thus, the idea of this proposal is to come up with a more objective set of requirements that measure effortless use of AV, and gather further evidence to support our vision of effortlessness.

We sum the proposal up with the following three main categories that are the *scope*, *integrability*, and *interaction techniques*. The idea is to apply all the categories to one single system to come up with an estimate of system effortlessness.

Before applying the proposed categories, our proposition is that the task should be defined *problem based*, thus leaving enough space for interpretation how to solve the problem with the current system. Different systems might provide different perspectives on how to tackle a problem. This is our starting point, and examples of such problem based tasks are, for example, anything between "demonstrating quicksort algorithm" and "teaching sorting algorithms", but not "how to animate quicksort partition algorithm" (as the target system might have some other alternative than animation to visualize the partition algorithm). Only after we have agreed the context, it should be possible to evaluate and compare different systems *that are targeted to such activity*.

Category Scope is basically defined in how wide context one can apply the system. The results show clearly that the lack of time is the most common reason for not using AV as widely as expected. Thus, a system providing a more broader context to apply is definitely more attractive choice than many single purpose systems because it saves the time to install and learn various tools that all can be applied only for a very limited scope. This can also be seen from the responses for the question about *ideal tool*. The respondents had a vision of "general tool" that combines several small tools together or comes true in several tools that have similar usage (see, e.g., Table 1). Moreover, in Section **??**, we can see that in the free responses the scope of the creative use is not just a single specific topic, but a broader context.

The scope can have the following subcategories defined, for example, in terms of the subdivisions of a course: lesson-specific, class-specific, domain-specific, and generic (not any domain specific). A generic tool, however, can still be, for example, lesson-specific: the tool provides tutorials and ready-made examples for some specific lesson (e.g., sorting algorithms). The deeper the system covers the subcategories the better. The respondents of the survey seem to be keen on downloading not only tools but also something they can use for their class (e.g., data structures and algorithms), thus there is also need for good tutorials as well as ready-made examples. Even if the examples of a generic tool do not contain the desired learning objects, they should help one to figure out how to apply the tool to produce more content. Similarly, a generic tool can have very small scope if there is no evidence that it will be used for other domains (e.g., computer science) as well. A good indication that a system has large scope is that a third party can produce content with it.

Category Integrability covers topics such as easy installation, but it is a much more broader category than that. We also include features such as customization of visualizations, platform independence, internationalization, good tutorials, etc. Thus, this category lists the features that makes the system attractive to use because there is a way to integrate the system into a course. No single feature makes the system effortless in this sense, but together they have to provide a meaningful way to make the system applicable. It is even better if there are several ways to reach the goal as there are many content generation approaches that meet with support (See Table 3).

Subcategories for integrability can be defined in terms of desired features such as described, for example, in Rößling and Naps (2002). They rank the systems (among others) according to the following requirements: interactive prediction support (e.g., stop-and-think questions or algorithm simulation exercises), database support (for course management), and integration of hypertext. The list is not exhaustive, but gives a hint what kinds of features are expected. If there are many feasible features available, it means more ways to complete a task. Moreover, the more ways there are to achieve the goal, the more effortless the use of the system is. Ultimately, integrability is measured in terms of how well a third party can adopt the system to their course.

Category interaction techniques. Even though a system has a very large scope, and it is easy to integrate it to a course, it may lack this very important aspect. The trend is towards activity where more interaction is required. Table 2 shows that there are many tasks that all seem to be relevant at least to some extent. However, in order to cover as many of these as possible, a tool must support more interaction than what comes with a simple VCR type of animator panel that has backward, forward, and play buttons. This is true with production of material for teaching (towards dynamic on-line illustrations instead of static lecture illustrations) as well as production of material for practice sessions (towards student interaction instead of tutor-centered demonstrations). It is not only important that the interaction between the system and visualizer (content creator) be adequate, but the system has to support interaction between the visualization and the end user as well. Here the end user might be, for example, a learner that solves exercises.

Interaction techniques have two subcategories, which are the two point of views described above: *producer* and *consumer*. Both point of views share some common interests, and we could measure the systems in terms of usability. We do, however, not discuss this or other human computer interaction (HCI) issues here any further. The taxonomy of Price et al. (1993), for example, suggest categories to measure interaction methods, but as we have argued in our previous study (Karavirta et al., 2002) they usually do not cover the various later interaction methods very well. For example, it is very hard to categorize Matrix (Korhonen and Malmi, 2002) that promotes visual algorithm simulation, a technique that allows the user to interact with the underlying data structures in terms of direct manipulation, in the taxonomy. Thus, instead of trying to rule out all the possible interaction techniques at the moment and in the future, we look at the subcategories by determining how well they correspond to the required uses cases (see, *e.g.*, Table 2).

Table 3 implies that there is a number of content generation approaches a single system can support, all of which were preferred by many of the respondents. Interestingly, the most cited technique *interactive simulation* is the one that requires the most interaction. Moreover, each approach requires possibly different interaction techniques. Thus, in general, the more approaches a system supports the better. Or other way around, the more interaction techniques (what ever they are) the system supports, the more content generation approaches can be achieved with it. The question is, can we cover the task with the tool or not, *i.e.*, does the system support the required interaction techniques involved? However, the question how well does systems support an approach is left to be measured in the other categories.

Even though the point of view in our survey was the producer's perspective, we cannot neglect the consumer who eventually uses the produced visualizations. Thus, we have to look

into the interaction methods also from this point of view. We are not, however, interested in the "can we cover" question anymore, but instead in how well does the interaction techniques involved support the various learning strategies. There exist several indicators (see, *e.g.*, Kolb (1984); Felder (1996); Bloom (1956); Naps et al. (2003a)) that can be used to measure such things depending on the purpose of the evaluation. For example, one can apply the *engagement taxonomy* introduced in Naps et al. (2003a) to evaluate the level of activity in the learner–system communication. The levels introduced are viewing, responding, changing, constructing, and representing. A single system can support any combination of these, and the more levels it supports the better. The effortlessness comes from the fact that as our knowledge on how to support learning increases, we might want to change the level of activity, but not the tool. If the tool supports many levels, there is a better chance that we can continue to use it even though the requirements change. Moreover, several activity levels might be attractive in a single course in order to be able to apply the same tool for several levels (e.g., viewing in lectures, and changing in practice session).

5 Conclusions

In this paper, we have presented the results of the on-line survey conducted prior to the PVW'04 conference. In addition, based on the results, we have proposed a taxonomical approach to measure the effortlessness of algorithm visualization systems. The taxonomy contains three main categories that are the *scope*, *integrability*, and *interaction techniques*. The categories try to characterize the evaluated system by determining the extent of its scope by answering whether a third party can produce content with it; integrability by measuring how well a third party can adopt the system to their course; and interaction techniques by looking at the system from producer's and consumer's perspective, and by determining how well the system corresponds to several common use cases.

The survey had a smaller sample set compared to other similar surveys carried out in the past five years. In addition, too few non-developers answered the survey and that might have biased the results. However, we still believe that we have managed to gather data that reliably supports our view of effortless AV creation.

One concern that raises up from the survey data was the observation that respondents are keen to use AV quite *passively*, *i.e.*, the instructor is active with the tool and the learner is only passively viewing the visualization. In our opinion, however, the use of AV has much more potential than that. In this sense, we must pay attention to the *interaction techniques* supported by the AV tools since as long as there are no suitable systems available, the use of AV retains its passive form.

Finally, it seems that the instructors have more time available to search AV tools than they actually use. Thus, developers should pay attention to ease the system integration as well as to promote the system use for different contexts.

In the future, our aim is to define a full taxonomy for effortlessness in algorithm visualization by applying it to different use cases (producer's perspective) based on this work. Some of the existing algorithm visualization systems will be classified based on the use cases they are suited for. We strongly believe that this kind of classification would help us design better algorithm visualizations systems for different needs and increase users ability to select correct tool for a specific problem. Also, when compared to e.g. Price's Taxonomy (Price et al., 1993) and applied on several systems this would give us knowledge about the consequences of different design selections.

Acknowledgments

We thank Guido Rößling and other participants of PVW 2004 who commented the questionnaire and earlier versions of this paper.

References

Ronald M. Baecker. *Sorting Out Sorting: A Case Study of Software Visualization for Teaching Computer Science*, chapter 24, pages 369–381. The MIT Press, Cambridge, MA, 1998.

Benjamin S. Bloom. *Taxonomy of Educational Objectives, Handbook 1: Cognitive Domain*. Addison Wesley, 1956.

John Domingue. Software visualization and education. In Stephan Diehl, editor, *Software Visualization: International Seminar*, pages 205–212, Dagstuhl, Germany, 2002. Springer.

Richard M. Felder. Matters of style. *ASEE Prism*, 6(4):18–23, 1996.

Jyrki Haajanen, Mikael Pesonius, Erkki Sutinen, Jorma Tarhio, Tommi Teräsvirta, and Pekka Vanninen. Animation of user algorithms on the Web. In *Proceedings of Symposium on Visual Languages*, pages 360–367, Isle of Capri, Italy, 1997. IEEE.

Christopher D. Hundhausen, Sarah A. Douglas, and John T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3):259–290, June 2002.

Ville Karavirta, Ari Korhonen, Jussi Nikander, and Petri Tenhunen. Effortless creation of algorithm visualization. In *Proceedings of the Second Annual Finnish / Baltic Sea Conference on Computer Science Education*, pages 52–56, October 2002.

David A. Kolb, editor. *Experiential Learning: Experience as the Source of Learning and Development*. Prentice-Hall Inc, New Jersey, USA, 1984.

Ari Korhonen and Lauri Malmi. Matrix — Concept animation and algorithm simulation system. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 109–114, Trento, Italy, May 2002. ACM.

Paul LaFollette, James Korsh, and Raghvinder Sangwan. A visual interface for effortless animation of C/C++ programs. *Journal of Visual Languages and Computing*, 11(1):27–48, 2000.

Fernando Naharro-Berrocal, Cristobal Pareja-Flores, Jaime Urquiza-Fuentes, J. Ángel Velázquez-Iturbide, and Francisco Gortázar-Bellas. Redesigning the animation capabilities of a functional programming environment under an educational framework. In *Second Program Visualization Workshop*, pages 59–68, HornstrupCentret, Denmark, 2002.

Thomas L. Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodgers, and J. Ángel Velázquez-Iturbide. Exploring the role of visualization and engagement in computer science education. *SIGCSE Bulletin*, 35(2):131–152, June 2003a.

Thomas L. Naps, Guido Rößling, Jay Anderson, Stephen Cooper, Wanda Dann, Rudolf Fleischer, Boris Koldehofe, Ari Korhonen, Marja Kuittinen, Charles Leska, Lauri Malmi, Myles McNally, Jarmo Rantakokko, and Rockford J. Ross. Evaluating the educational impact of visualization. *SIGCSE Bulletin*, 35(4):124–136, December 2003b.

Blaine A. Price, Ronald M. Baecker, and Ian S. Small. A principled taxonomy of software visualization. *Journal of Visual Languages and Computing*, 4(3):211–266, 1993.

Guido Rößling and Thomas L. Naps. A testbed for pedagogical requirements in algorithm visualizations. In *Proceedings of the 7th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'02*, pages 96–100, Aarhus, Denmark, 2002. ACM.

John T. Stasko. Using student-built algorithm animations as learning aids. In *The Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education*, pages 25–29, San Jose, CA, USA, 1997. ACM.