OpenDSA: Beginning a Community Active-eBook Project

Clifford A. Shaffer
Dept. of Computer Science
Virginia Tech
shaffer@cs.vt.edu

Ville Karavirta
Dept. of Computer Science
and Engineering
Aalto University
vkaravir@cs.hut.fi

Ari Korhonen
Dept. of Computer Science
and Engineering
Aalto University
archie@cs.hut.fi

Thomas L. Naps
Dept. of Computer Science
University of Wisconsin,
Oshkosh
naps@uwosh.edu

ABSTRACT

In this paper, we present our vision for OpenDSA, an open-source, community-based effort to create a complete active-eBook for Data Structures and Algorithms courses at the undergraduate level. We define active-eBooks as going beyond classic hypertextbooks, being a close integration of text and images with interactive visualizations/simulations and assessment activities. The OpenDSA project is meant to proceed with broad participation from the CS Education community, with maximum flexibility on reuse of materials, and with the ability for a given instructor to pick and choose material from the collection and modify as desired. We discuss the goals of the project, our initial community organization efforts, and the technical infrastructure that we envision for the project. Initial progress is described.

1. INTRODUCTION

The field of Algorithm Visualization (AV) has made steady progress with more AVs available and better pedagogical studies helping us to understand how to make best use of them [17]. Yet, actual AV use in the classroom is still low compared to AVs' claimed favorable view among instructors. Survey results over a number of years have consistently shown that instructors face many impediments in using AVs: difficulty finding good materials, lack of time in classes or lack of time to change the classes, and lack of understanding on how to make best use of them [9, 16].

Good online course materials that can provide complete instructional support for a data structures and algorithms class could solve many of these problems. If the resources were well recognized and supported within the CS community, then instructors could feel confident in using such material. As a complete solution for instructional material, it could replace an old implementation for a class in much the same way that instructors adopt traditional textbooks and coursenotes. This approach solves problems that instructors report as a major impediment when "squeezing in" an AV that does not fit well with the existing instruction process. At the same time, online interactive courseware can hope to improve exposition through a richer collection of technologies than are available through print textbooks, and increase student engagement with the material, allowing them to learn at a higher level in Bloom's taxonomy [9].

Fortunately, a number of recent technical advances improve the feasibility of developing, deploying, and using interactive courseware. Increased access to the Internet by students and instructors, both in and out of the classroom, makes AV use more practical. This makes a huge difference in the confidence of mainstream instructors for using such technology, in contrast to early adopters [3]. Another potential factor in favor of the use of AVs and interactive courseware is ubiquitous availability of laptops and mobile devices. However, there is a downside to the non-PC devices, in that they place various technology limits on delivering content. For example, Java Applets and Flash cannot be displayed on most such devices.

We will use the term *active-eBook* to refer to courseware that goes beyond the classic conception of a hypertextbook. It closely integrates text and images with interactive visualizations, simulations, and assessment activities. See [10, 11] for background on efforts to define and implement such artifacts.

Data Structures and Algorithms as a topic can particularly benefit from the use of advanced technology to aid explanation of the dynamic processes that make up the essence of an algorithm, and which can be difficult to convey using words and images. There exists no complete electronic textbook, tightly integrated with AVs, that could be used as the primary learning resource in a semester-long computer science course in data structures and algorithms. This is perhaps surprising because Marc Brown's groundbreaking dissertation on AV [1] issued the following caution:

> Much of the success of the BALSA system at Brown [at the time Brown's thesis was written]

Koli Calling '11, November 17–20, 2011, Koli, Finland.

> is due to the tight integration of its development with the development of a textbook and curriculum for a particular course. BALSA was more than a resource for that course – the course was rendered in software in the BALSA system.

Why have CS educators not heeded Brown and authored active-eBooks? We suspect the answer is something known to anyone who has written a textbook or an AV: it consumes huge amounts of time. While writing a textbook is a big job, writing an associated set of AVs and the assessment support is a far bigger job yet.

Thus, the most practical way to achieve the goal of a complete active-eBook is to involve a broad community of developers in the effort. This calls for an open-source development effort, with the maximum ability to reuse materials.

In this paper, we discuss our plans to create an active-eBook for a complete semester course in data structures at the Sophomore level. Our particular focus is on the use of algorithm visualization [9, 17] as a means both to deliver the necessary dynamic exposition, and to increase student interaction with the material.

2. IMPLEMENTATION PRINCIPLES

When considering the full breadth of content that would be contained in a complete course textbook, we conclude that three distinct forms of content presentation are desirable. First, no matter how dynamic and interactive the topic, text and images as are now found in typical textbooks continue to have their place as part of the exposition. Some content simply is not visual or dynamic and so is efficiently transferred via words and images.

Second, some content is essentially expository (i.e., at that point in the presentation there is no need for student constructive interaction), but the content is about dynamic processes or conducive to visual presentation. This includes most algorithm descriptions, such as how a particular sorting algorithm works. Since initial presentation does not involve exploration or decision making, or demonstration of proficiency, the prime concern is what techniques provide the clearest explanation. This could best be handled by a presentation that relies heavily on diagrams and simple animation, with pacing controlled by the the reader. In short, an animated slide show is adequate for such presentations.

Third, there exist a vast number of instances where the presentation would be improved by providing student interaction. This includes things as simple as probing a calculation, (e.g., trying different inputs to a simple simulation or equation). An example is the famous Birthday Problem: How many people need to be in a room before the odds are greater than even that two share a birthday? Instead of presenting a statement of the answer, or even an equation to compute it, a simple interaction allows the student to give a value for number of students. The response is the result of the calculation (the probability of a shared birthday). This gives some small opportunity for exploration. An active-eBook full of such opportunities can permit far greater engagement.

A deeper form of interaction comes when demonstrating proficiency with an algorithm, such as the interactive exercises for tree insertions that are part of TRAKLA2 [6, 7]. Here, students must drag-and-drop nodes at the correct position in the tree. These are typically handled by something like a Java Applet and dynamic on-the-fly calculations/processing of the associated algorithm. Performance comparisons provide another opportunity for engagement. Instead of simply claiming that one approach has better performance than another, we can invite the student to provide inputs and run built-in simulations of multiple data structures or variants to see how they perform.

Binding all of this together should be a steady stream of assessment activities to make sure that students stay "on track" and to keep them engaged even during otherwise passive exposition. This can be done with simple pop-up questions (that might or might not require a successful response to continue) and end-of-section quizzes whose success might be required to demonstrate competence needed to continue on to the next section.

As we progress from the first to the third of these presentation approaches, the development cost goes up dramatically. Text and images can be developed relatively quickly. In contrast, it took several student-years of effort (and over two actual years) to develop the Virginia Tech Hashing Tutorial (`http://research.cs.vt.edu/AVresearch/hashing`). This is a complete unit of instruction that corresponds to about one week of class. These development costs came mainly due to the effort involved in developing a handful of Java applets. A properly animated slideshow takes longer to create than equivalent text and images, but should be significantly faster to implement than a fully interactive exercise.

3. TECHNICAL CONSIDERATIONS

A number of technologies are available for developing each of the three presentation types described in the previous section. Text and images can certainly be done with any traditional web development tool. The second component, which we characterize as an animated slide show, can be done (as the characterization suggests) using a variety of presentation tools such as MicroSoft Powerpoint, LaTeX's Beamer package, OpenOffice Impress, or Apple Keynote. However, while presentations can be created in these tools, the resulting presentations cannot so easily be integrated with the rest of the active-eBook. Browsers can support some of these tools as plugins, but not universally across a range of devices. The presentations can be converted to PDF format, but only Adobe's Reader can actually display the animations (at least, evince and xpdf, popular alternative PDF readers do not). Nor do the PDFs well integrate with non-PDF portions of the whole, or with embedded assessment activities.

Java and Java applets are currently the most popular choice when it comes developing AVs. However Java faces significant compatibility difficulties with various browsers and mobile devices, difficulties that are becoming greater over time rather than improving. Flash is another popular tool for developing animations, and could support the interactive activities as well. However, Flash requires a plugin on most browsers, and so is not compatible with devices such as the iPad.

One technology robust enough to implement all desired dynamic and interactive components is HTML5 incorporating JavaScript and CSS. HTML5 integrates its dynamic components well with standard text and images, and easily ports between PC browsers and mobile devices. Potential concerns include ease of use for content developers (as compared to, for example, PowerPoint when developing ani-

mated slide shows), and level of penetration of the necessary browser technology. Given that alternatives such as Flash are at least as problematic in terms of a typical user having access (since they require plugins), the problem of penetration seems to be low and quickly receding. College-level students tend to have access to moderately up-to-date browser technology. And the core JavaScript/CSS technology is well supported by browsers, even if some more advanced HTML5 components are not. Thus, we currently advocate use of HTML5 technology for developing the dynamic components of the active-eBook. We provide more details in Section 6.

4. INTEGRATED ASSESSMENT AND MONITORING

There is much evidence that AVs foster effective learning when presented in a way that forces the student to actively engage with the visualization instead of passively viewing it [4, 9]. Additional engagement can be created by having the student respond to questions or demonstrate proficiency. Although many AVs include questions, few do so in a way that allows an instructor to monitor their students' progress. More typically the student's interaction with the system produces immediate feedback to the student, but the assessment of that interaction is not recorded in a way that the instructor can access. Nor do the students' answers provide a persistent record for the student that a section has been mastered, or integrate with navigation through the content such as provided by an "intelligent tutor". Two AV systems that do support this sort of integrated assessment are TRAKLA2 [6, 7] and JHAVÉ [8]. Although TRAKLA2 and JHAVÉ support online assessment of students' use of visualizations, they do so in unique, non-portable ways. We seek an approach that can work with a variety of presentation mechanisms as well as the active-eBook structure itself (some but not all questions will come within AVs).

One possibility is a decoupled approach to assessment, where the assessment process is done by following a link to a third-party site that provides the relevant series of questions. This might take place at the end of each section in the textbook. But a strongly decoupled approach does not support assessment activities from inside dynamic activities, such as TRAKLA2-style proficiency exercises. A more flexible approach still involves third party sites, but they are structured to provide web services. This would even allow the assessment infrastructure to be broken into separate services for (i) supplying questions from a bank or automated generator, (ii) evaluating the student answers, such as for a programming question, and (iii) collecting and managing the students' solutions for the instructor.

Among the factors that will have to be considered in developing the assessment architecture are:

- Developing effective strategies for evaluating student responses. Assessing multiple-choice, multiple-selection, and fill-in-the-blank questions is straightforward. But automated assessment of textual responses is clearly much more difficult. Automated assessment of the correctness of a programming exercise is also possible with the right infrastructure.
- Supporting rich activities specific to CS. This includes multiple-choice questions dependent on the run-time conditions of an algorithm, visual algorithm simulation exercises such as in TRAKLA2, and small programming tasks to be evaluated by comparing output from the proposed solution to the answer key.
- How to represent test questions. There exist standardized question representations, such as the IMS Question and Test Interoperability specification (QTI).[1]
- How to make all of the above a little bit different for each student or trial so that copying answers from each other is not possible, and the same exercise can be redone to get more practice.
- How to authenticate the student, and store the responses and results in a way that makes it easy for instructors to analyze their students' progress.
- How to store assessments of students' work in a way that respects their privacy.
- How to "loosely couple" the assessment system with the active-eBook. Because the visualizations will be launched out of the active-eBook we will need to have a mechanism for recognizing the learner's identity so as to interact with the quiz/assessment database that might be stored on a different server.

5. THE CREATIVE COMMONS

Next we consider the broader context in which development of the active-eBook should take place. Such a project is a huge undertaking. As evidence of this (besides our own experiences with the extraordinary amount of time that it takes to develop high-quality AVs), consider that there exist few examples of the type of artifact that we seek to create. Ideally, a broader community can be encouraged to contribute to the project, much in the style of an open source software development effort. The authors have many collaborators within the broader AV community, and ideally we can leverage these collaborations to develop the materials. Since we envision the materials to be distributed with a GPL or Creative Commons license, intellectual property rights will be less of an issue than if a commercial publisher were involved.

The Connexions Project (`http://cnx.org`) is presently the largest collection of online textbooks developed with a creative commons license model. Connexions is more than a collection of publicly available online textbooks. They use a "creative commons" infrastructure that makes it easy for authors to reuse and combine pieces of textbooks, or to make their own altered version of an existing textbook. We envision developing an active-eBook within such an infrastructure to support sharing and reuse. There are integration concerns related to, for example, HTML5 technology or other technology for developing dynamic presentations. Integration of assessment is also of concern, but Connexions has recently begun developing support for assessment.

Other collaborative commons for online learning materials exist. For example, LeMill (`http://lemill.net`) is a Web community for finding, authoring and sharing educational resources supporting also social media features and many different languages. Many LeMill-based communities have evolved around learning materials either on specific topics or written in certain languages.

One of our architectural concerns, especially with a large number of contributers and potential users involved, is flexibility. It should be possible to access the material from

[1] `http://www.imsglobal.org/question/qtiv1p2/imsqti_oviewv1p2.html`

multiple platforms, such as LeMill and Connexions. This increases the desirability to decouple the various architectural components (such as text/graphics) from dynamic components from assessment services.

We envision a multistage process to develop the active-eBook project. The first step is to devise a complete management plan and to define the development workflow within the chosen implementation infrastructure. Next is to define a detailed "storyboard" defining a detailed layout for at least a complete semester's worth of topics, with all of the text and detailed descriptions of all places where interactive activities, assessment, or dynamic presentation are desired. These detailed descriptions could include mock-ups in, for example, PowerPoint, or pointers to existing AVs. Third is to initiate an open development process where submissions from interested parties are provided for specific activities called for in the Storyboard, coupled with a reviewing process. If developed in this way, the active-eBook will become "owned by" a broader community of CS educators.

6. THE JSAV LIBRARY

For a number of reasons, the OpenDSA project will benefit from providing development tools for AVs and other dynamic components of the system such as interactive exercises and simulations. First, having such a library makes it easier for others to contribute resources since they can develop content more easily. There currently exists little AV support for JavaScript. Second, using such a library helps to make the various dynamic components more consistent in look and feel. Third, providing the library and corresponding documentation helps to encourage consistent best practice in AV development. We have begun experimenting with various approaches to implemention based on JavaScript, and call the resulting library JSAV for JavaScript AV.

Several general AV systems have been developed in the past, with nearly all of the systems now active based on Java. Many popular ones use a scripting language, including Animal [14], JHAVÉ [8], and ALVIE [2]. Required features for pedagogically effective AVs have been studied by many [12, 13, 15]. Learning from this prior work, the following summarizes our requirements and ideas for how the library will support them.

Flexibility: Our vision of the library will include data structures and automatic layouts and rendering of them along with graphical primitives that can be used to add "decorative" elements to the visualization. Since the aim is to support a wide range of topics and have multiple visualization designers, it is important that the library is both flexible and extensible. Thus, it should also be possible to add new data structures, new layout functionality for existing structures, and other extensions to the library with reasonable effort.

Overriding existing functionality must be possible. For example, the visual representation of a "swap" operation in a sorting AV should be something provided by the library, but also replaceable by a given developer. As another example, one developer might wish for a sorting algorithm to show an array as a horizontal series of boxes with numbers in them. Another developer might want a horizontal series of vertical bars with the value of each bar underneath, but no visible array boxes around these numbers.

Visualization Control: A student should be able to control step by step visualization execution, going both forward and backward. Additional controls should include changing the speed of animated sections, or playing the entire slideshow as an animation. Most presentation-oriented AVs take the form of a "post mortum" dump of a sequence of states and their visual transitions, with the ability to move forward and backward through the sequence (using "standard" videoplayer controls). Even an AV that seems to be working on-the-fly can often make use of the state sequence paradigm between key stages. For example, consider a search tree AV that lets users enter values to insert/search/delete. For any given operation on the tree, the user can enter the value, and the AV can generate the sequence of states to perform that operation. Then the user can step back and forth through a visualization of the sequence. Implementing this functionality requires storing the states and forward/backward transitions between them.

Explanations: AVs should include textual explanation of the action that is taking place, since dynamic messages have been shown to be one of the most important features of effective AVs. These comments are generated by the AV as the visualization progresses. Often this involves computation of values tied to the particular input example being processed. If such input is generated on-the-fly, then obviously the contents of the messages also have to be generated on-the-fly, or at least parameterized. In addition, messages need to be color coded.

Orthogonal to defining the messages themselves is the delivery of the messages onto the screen. There are different ways that this can be done effectively. One approach involves a window, usually taller than it is wide, that gives room for many messages. A scroll bar allows the user to see previous messages. However, sometimes a designer will want another approach if, for example, it is not viable to give this much screen real estate to the messages. Or perhaps a history is not needed, and showing only one message at a time is preferred. The messaging system must be flexible to adapt to various needs. It should be possible to define a message pane with scroll bar, or to define a simple one-or-two line overwriting scheme simply by changing configuration parameters when initializing the visualization. Even more flexibly, any component on the page might register as an event listener for new messages and display them as it sees fit.

Engagement: Engagement is the key to effective AVs [4, 9]. A simple form of engagement can be created through pop-up questions such as are available in JHAVÉ or Animal. Custom input data provided by the user also forces engagement. Greater engagement comes from students simulating the steps of an algorithm such as in TRAKLA2. In our vision, the library will provide support for all of these engagement methods. Like the messages, pop-up questions related to the content of the visualization will need to be generated on-the-fly when the student is using the visualization. Again, the display of the questions needs to be flexible to suit various screen sizes and visualization layouts.

TRAKLA2 like visual algorithm simulation [5] requires a different approach. The library needs to support specifying operations that happen in the data structures when the student clicks or drag-and-drops components within the visualization. The result of these operations should be a visualization sequence. In addition, the library should support sending a trace of operations/states to an assessment

server (see discussion in Section 4), or alternatively, grade the assignment in the browser.

Pseudocode: Pseudocode display is a common feature of AVs. Most developers want to provide this functionality in the system if it is not too much work. But building the infrastructure is often considered too difficult for many developers. Therefore we should supply this, and do it right. The best pseudocode system that we know of appears in Algorithms in Action (AIA) [18], and we want our library to support something similar. The AIA approach to pseudocode display has the concept of collapsing hierarchies, where a function might be presented either by a single comment or by multiple lines, each of which might in turn be expanded into multiple lines. Once this relationship is specified, everything should be handled automatically for the developer aside from actually calling the highlighting function in the library that indicates the "current" pseudocode line to be highlighted at any given instant in the visualization. Of course, the pseudocode drives the AV as well. When a section of the pseudocode is closed, the visualization should skip over those details. Thus, the AV driver has to be able to get the necessary information about the state of the pseudocode hierarchy.

We imagine this to work by having a replaceable module within the library that can be queried by the visualization that will return information whether a certain step should be visualized. One implementation of the module might get the information from the folding of the pseudocode. Another implementation might query a student knowledge model (when used with an assessment server) to decide whether the step should be shown to the student.

Server-side Support: To be usable in grading, the AV library needs to connect to an assessment system to get a grade and to a course management system to store the results. Communication can be done using AJAX, but as discussed in Section 4, the specification for the questions, answers, and grades is still undecided.

User Interface: There typically are many distinct visual components to an AV, including possibly pseudocode, messaging, and one or more visualization panes. CSS styling will allow a high degree of tailored configurations for the AV container to define various panes as desired. For example, one can define the messaging pane of arbitrary size and position within the container, with options as described above regarding scrolling vs. overwriting. Messages automatically get sent there. One can define a pane of whatever size and position, which automatically presents the associated pseudocode. One can define a main visualization pane, and direct graphical objects to there. Or two visualization panes, and send appropriate objects to each. As the distinct components of the AV will be HTML, the layout of the panes can be flexibly specified using standard CSS styling to change the default layouts of the library.

Ideally, JSAV will support all of these requirements. In addition, there are JavaScript and browser environment-related requirements. The library needs to support having multiple visualizations on one browser page. Furthermore, the layouts of the data structures should adapt to the browser window size (including mobile devices). Finally, it should support an event-based architecture for communication between different components. This will enable loose-coupling of components, and facilitate the introduction of other components offering similar but customized functionality as the ones in the library. As a simple example, this could be listening for message events to show them as pop-up alerts instead of changing the contents of an HTML element.

The development of the library has already started. Currently, the library prototype supports several of these requirements. The user interface can be specified with HTML, and JSAV functionality such as messages/explanations and visualization controls can be added with HTML class attributes. Data structure array is supported with multiple layouts and effects to modify the array. Engaging visual algorithm simulation exercises can also be created. The assessment of such exercises is done within the browser in two possible modes: continuous feedback after each step and on request.

7. CONCLUSIONS AND FUTURE WORK

We aim to build a community that will commit to developing an active-eBook covering many CS topics, including many high quality interactive AVs and many assessment exercises. This will require many participants. In addition, we need an infrastructure to maintain the materials and to allow data flow among different components. The focus is on rich types of activities beyond the current state-of-the-art.

A project this large requires that those who work on it have input into the development process early and often. Otherwise decisions will be made that do not reflect a broad perspective, leading to a rigidity in standards that might discourage participation from those who might otherwise be interested. This early input must come both from those who see as their main contribution the authoring of textual material that employs AVs and from those who see as their main contribution the development of software components of the JSAV library. Decisions to be made by those who will be authoring textual material include the nature of the pseudocode that will be used to describe algorithms and the style of mathematical notation used to analyze algorithm efficiency. Those who will contribute their efforts to the JSAV library need a unified vision of the APIs they will use for the variety of subsystems that will comprise the library. These include APIs for:

- Data structure support
- Graphical primitive support
- User controls
- Messages/explanations
- Pseudocode display
- Question-creation API
- Input generator API
- Algorithm simulation API
- Assessment API

We have developed example tutorials that point the way toward the desired goal. In addition to the aforementioned Hashing Tutorial, the AlgoViz catalog includes, for example, a Binary Heap Tutorial embedded with TRAKLA2 exercises. Moreover, we have experimented with TRAKLA2 and JHAVÉ, interconnecting the two systems together.

Many management issues need to be resolved. These include how and where to store all materials in a way that supports open source-style development. Our initial efforts at developing JSAV are already available at the OpenAlgoViz SourceForge project repository for public inspection. But this might not be sufficient to support a larger community effort.

We also need a reviewing mechanism for dealing with accepting and integrating contributed material. The material should be peer reviewed and improved based on the feedback. Feedback should be in terms of comments suggesting improved wordings or versions (in case of text and figures) or tasks describing how to improve the AV (in case of software). A project manager for each chapter could be responsible of making the decisions which changes to accept or reject.

Further information about the status of the OpenDSA project can be found at the community forum (`http://algoviz.org`) and the project Wiki (`http://algoviz.org/ebook`).

As this is a discussion paper, we raise a number of questions for the larger community (and the delegates of Koli).

- Do you agree with our vision of the open data structures and algorithms active-eBook?
- How can it be refined?
- Are you interested in participating in such a project?
- How should we inform the rest of the community to pay attention to this project?
- How can we engage the community to participate in this project?

8. REFERENCES

[1] M.H. Brown. *Algorithm Animation*. MIT Press, Cambridge, Massachussets, 1988.

[2] P. Crescenzi and C. Nocentini. Fully integrating algorithm visualization into a CS2 course: A two-year experience. In *Proceedings of the 12th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, pages 296–300, 2007.

[3] K. Hew and T. Brush. Integrating technology into K12 teaching and learning: current knowledge gaps and recommendations for future research. *Educational Technology Research and Development*, 55:223–252, 2007.

[4] C.D. Hundhausen, S.A. Douglas, and J.T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, 13:259–290, June 2002.

[5] A. Korhonen. *Visual Algorithm Simulation*. Doctoral dissertation (tech rep. no. tko-a40/03), Helsinki University of Technology, 2003.

[6] L. Malmi, V. Karavirta, A. Korhonen, J. Nikander, O. Seppälä, and P. Silvasti. Visual algorithm simulation exercise system with automatic assessment: Trakla2. *Informatics in Education*, 3(2):267–288, September 2004.

[7] L. Malmi and A. Korhonen. *Active Learning and Examination Methods in a Data Structures and Algorithms Course*, pages 210–227. Number 4821 in LNCS. Springer-Verlag, 2008.

[8] T.L. Naps. Jhavé: Supporting algorithm visualization. *IEEE Computer Graphics and Applications*, 25:49 – 55, September 2005.

[9] T.L. Naps, G. Rössling, and nine more authors. Exploring the role of visualization and engagement in computer science education. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, pages 131–152, 2002.

[10] R.J. Ross and M.T. Grinder. Hypertextbooks: Animated, active learning, comprehensive teaching and learning resources for the web. In S. Diehl, editor, *Software Visualization*, pages 269–284. Springer, 2002.

[11] G. Rössling, T. Naps, and nine more authors. Merging interactive visualizations with hypertextbooks and course management. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, pages 166–181, 2006.

[12] G. Rößling and T.L. Naps. A testbed for pedagogical requirements in algorithm visualizations. In *Proceedings of the 7th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE'02*, pages 96–100, Aarhus, Denmark, 2002. ACM Press, New York.

[13] G. Rößling and T.L. Naps. Towards intelligent tutoring in algorithm visualization. In *Second International Program Visualization Workshop, PVW'02*, pages 125–130, Aarhus, Denmark, 2002. University of Aarhus, Department of Computer Science.

[14] G. Rößling, M. Schüer, and B. Freisleben. The ANIMAL algorithm animation tool. In *Proceedings of the 5th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, pages 37–40, 2000.

[15] P. Saraiya, C.A. Shaffer, D.S. McCrickard, and C. North. Effective features of algorithm visualizations. In *Proceedings of the 35th SIGCSE technical symposium on Computer Science Education, SIGCSE'04*, pages 382–386, New York, NY, USA, 2004. ACM.

[16] C.A. Shaffer, M. Akbar, A.J.D. Alon, M. Stewart, and S.H. Edwards. Getting algorithm visualizations into the classroom. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE'11)*, pages 129–134, 2011.

[17] C.A. Shaffer, M.L. Cooper, A.J.D. Alon, M. Akbar, M. Stewart, S. Ponce, and S.H. Edwards. Algorithm visualization: The state of the field. *ACM Transactions on Computing Education*, 10:1–22, August 2010.

[18] L. Stern, H. Søndergaard, and L. Naish. A strategy for managing content complexity in algorithm animation. In *Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education*, ITiCSE '99, pages 127–130, New York, NY, USA, 1999. ACM.