Open source software and the algorithm visualization community

Matthew L. Cooper\textsuperscript{a}, Clifford A. Shaffer\textsuperscript{b}, Stephen H. Edwards\textsuperscript{b,\textast}, Sean P. Ponce\textsuperscript{b}

\textsuperscript{a} Microsoft Corporation, Redmond, WA, United States
\textsuperscript{b} Department of Computer Science, Virginia Tech, Blacksburg, VA, United States

HIGHLIGHTS

- We review the open source practices used on algorithm visualizations (AVs).
- The state of the practice for open-source AVs is poor.
- Practical obstacles to open-source techniques on e-learning software are discussed.
- Solutions to these obstacles are presented.
- Community building and sharing will improve practices for e-learning developers.

ARTICLE INFO

Article history:
Received 1 May 2013
Received in revised form 16 December 2013
Accepted 30 December 2013
Available online 2 January 2014

Keywords:
Algorithm animation
Version control
Project hosting
Open source licensing
Open source tools

ABSTRACT

Algorithm visualizations are widely viewed as having the potential for major impact on computer science education, but their quality is highly variable. We report on the software development practices used by creators of algorithm visualizations, based on data that can be inferred from a catalog of over 600 algorithm visualizations. Since nearly all are free for use and many provide source code, they might be construed as being open source software. Yet many AV developers do not appear to have used open source best practices. We discuss how such development practices might be employed by the algorithm visualization community, and how they might lead to improved algorithm visualizations in the future. We conclude with a discussion of OpenDSA, an open-source project that builds on earlier progress in the field of algorithm visualization and hopes to use open-source procedures to gain users and contributors.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Algorithm visualizations (AVs) have a long history in computer science education, dating back to the 1980s [1]. AVs are widely viewed as having the potential for major impact on student learning [2–5]. However, studies on the pedagogical effectiveness of AVs are often equivocal [6], with some showing improved learning and others showing no significant difference. The primary determinant of pedagogical value appears to be related to the amount of interactivity and level of user engagement involved [6,2,3].

While many AVs have been produced, a high fraction of these are of low quality in terms of their usability, and overall utility [7]. We know of no studies that examine this issue directly, but we note that several of the systems recognized in [3] as being pedagogically effective had conducted usability studies. In contrast, it is reasonable to believe that a poorly
designed and implemented AV will also have low pedagogical effectiveness. This is one factor that might explain why some studies of AV pedagogical effectiveness are positive (they might be studying well designed and implemented AVs) while others show no significant difference when they are used (they might be studying less well designed and implemented AVs).

Nearly all AVs are distributed free on the Internet. A significant fraction (about half according to our examination of the AlgoViz catalog [8]) provide access to the source code for the project. Thus, AVs might seem to be a natural target for free and open source (FOSS) licensing and best practices, including use of development tools typically associated with the FOSS community.

In this paper, we report on the state of the practice for AVs regarding both development and distribution. To our knowledge, this paper presents the first attempt to focus on the development practices related to AVs. We find that, while many AVs are distributed with source code, they often do not qualify as open source. And nearly no developers of AVs report using open source development tools or best practices. Thus, we see much opportunity for improvements in the future. We report on one example of a project that is using open source practices, called OpenDSA [9–11]. OpenDSA builds on earlier progress in the field of AV development and hopes to use open-source procedures to gain both users and contributors.

Our fundamental argument extends from a well-known understanding within the software engineering community that better development practices lead to better software [12,13]. In the case of educational software development, adopting a set of standard development practices lets developers focus more of their attention on pedagogy and content development, rather than reinventing the wheel in terms of development practice. While the initial motivation for developing an AV might be for the developer’s own use (such as an AV developed by an instructor in order to teach their class), developers of such software typically also want others to make use of it. This must be the case for any developer who goes to the effort of posting their AVs in a public way. Within the universe of various development tools, FOSS tools in particular tend to provide mechanisms for better integrating with a community outside of the developer team, which helps both to improve visibility and to encourage outside participation. A desire for acceptance by a broader community can hopefully lead developers of AVs (and other educational software) to become more aware of similar software that already exists. As indicated in [7], many AVs developed in the past were unnecessarily duplicative of others. Better software practices, better community engagement, and greater community awareness should all help foster software that is both higher quality and more easily adopted.

The field of AV development practice might seem rather specialized (though AVs are becoming widely used, and there are many hundreds of them available). But we believe that AV development can be seen as representative of software development practices for other small-scale academic software development efforts, and so any results related to AV development should be more broadly applicable. We focus on AVs in this paper largely because we are most familiar with them and because we have adequate data available to draw reasonable conclusions. Since 2006, we have been tracking AV development and use [14]. Most of the statistics quoted in this article are based on the catalog of approximately 600 AVs maintained by the AlgoViz project [8].

When we say “open source”, we are specifically referring to software products licensed under an OSI (Open Source Initiative) approved license [15]. While other definitions of open source may merit consideration and be more or less inclusive, we have selected OSI because its definition of the term seems to reflect mainstream opinions on what an open license should and should not look like, and it gives defensible arguments for each of the conditions placed on such a license.

2. State of the practice

Since the mid-1990s and the advent of Java, nearly all AVs and AV development toolkits have been written in Java. While a few are written as stand-alone, one-off efforts by individuals, about 70% [7] are written as part of a collection, by a small team, perhaps within the context of a visualization development system. Thus, most AVs are developed under conditions that would seem to be good candidates for using professional development tools of some sort to manage the project.

2.1. Open source use in algorithm visualization

Nearly all algorithm visualizations that we are aware of are freely available for educators to use. It is possible that this will change in the future as textbook publishers and new companies like Zyante enter the market with commercial products that integrate visualizations with more traditional text content. But as of this writing, the AVs available are overwhelming free for use. That might be an artifact of our search process (primarily via online search using standard search engines), but the AlgoViz catalog contains only one non-gratis algorithm visualization or system out of nearly 600 separate visualizations that we have cataloged. Roughly 50% of these include source code online, and the number rises to about 60% when we include those who offer to make the code available on request.

Note that by “source code” we are referring to the source for the AV program itself. Many more AVs include code or pseudocode for the algorithm being visualized, which we do not consider as “source code” for the purpose of this discussion. The artifacts from our collection of AVs [8] were analyzed and separated into categories according to which open source license was chosen, if any. The categories are described in Table 1. Note that our analysis is at the level of an individual
Table 1
License types.

<table>
<thead>
<tr>
<th>License type</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPL-like</td>
<td>May redistribute the binary only if modified source available</td>
</tr>
<tr>
<td>BSD-like</td>
<td>May redistribute the binary with or without source; includes MIT license</td>
</tr>
<tr>
<td>Creative commons</td>
<td>Under some variant of the Creative Commons license</td>
</tr>
<tr>
<td>Public domain</td>
<td>Author explicitly relinquishes normal protections from copyright</td>
</tr>
<tr>
<td>Non-commercial</td>
<td>Non-OSI license or informal license that explicitly bars commercial use</td>
</tr>
<tr>
<td>Non-OSI</td>
<td>Declared open source under some other (non-OSI recognized) license</td>
</tr>
<tr>
<td>Unlicensed</td>
<td>Author makes source available but either: (1) makes no claim about redistribution, (2) makes a non-OSI-approved claim about redistribution, or (3) explicitly disallows redistribution</td>
</tr>
<tr>
<td>Contact author</td>
<td>Source code not online; author claims to make source available through private contact under unspecified terms</td>
</tr>
<tr>
<td>Source unavailable</td>
<td>Redistribution not an issue since source is unavailable</td>
</tr>
</tbody>
</table>

Table 2
Counts of AVs in the AlgoViz catalog by license type.

<table>
<thead>
<tr>
<th>License type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code unavailable</td>
<td>208</td>
</tr>
<tr>
<td>Contact author</td>
<td>45</td>
</tr>
<tr>
<td>Unlicensed</td>
<td>55</td>
</tr>
<tr>
<td>Non-commercial</td>
<td>53</td>
</tr>
<tr>
<td>Non-OSI</td>
<td>38</td>
</tr>
<tr>
<td>GPL-like</td>
<td>57</td>
</tr>
<tr>
<td>BSD-like</td>
<td>7</td>
</tr>
<tr>
<td>Creative commons</td>
<td>25</td>
</tr>
<tr>
<td>Public domain</td>
<td>2</td>
</tr>
</tbody>
</table>

Visualization. Many AVs come as part of a larger collection, and invariably all AVs in the collection will have the same licensing terms. Thus, the numbers might be considered skewed in favor of the larger collections.

Results of this analysis were somewhat disappointing from the FOSS standpoint. Of 490 visualizations in the catalog for which the license status is recorded, only 282 had source available (and only 237 actually posted source on the web, while the other 45 indicate availability via email). Of those 237, only 89 were under recognized licenses (GPL, MIT, BSD, CC, etc.) with another 92 under less formal open source licenses. Typically, these “informal” licenses explicitly disallow commercial use. For academic courseware, restrictions on commercial use are largely irrelevant, so our analysis will treat those artifacts as open source. Only 2 visualizations are explicitly stated to be in the public domain, while 55 of the visualizations post the source code with no license or other explanation of their conditions for use. See Table 2 for a full breakdown of the various categories.

We had hoped that more than 18% of the AVs would be truly open source in the sense of having recognized licenses. Of AVs providing source, those with recognized licenses jump to 38%, a better but still unimpressive number. There is a reasonable expectation that more AVs would be open source if there were greater awareness on the part of developers regarding the advantages of providing an explicit open source license with their source code. Many of the source-available AVs seem to lack a license not because the developers wish to prevent redistribution, but instead due to lack of understanding of the process. For instance, groups of AVs in the category “Contact author” are from the same developer, and since one would have to email him or her to get the source code, it seems likely that arrangements for redistribution could be worked out at that time. Also, at least one of the AVs in the “Source unavailable” category was used to create a derivative work by an independent party, so it seems that some kind of redistribution scheme was worked out on a case-by-case basis between the two groups.

2.2. The role of open source in AV development

So why should anyone care whether or not a given artifact is technically open source? Source code provided without an open source license is essentially useless to potential users of that source code. Suppose an instructor wants to fix a bug in or make a change to an AV. The instructor is free to make those changes for herself. However, because of copyright law, she probably cannot redistribute the work. That is, copyright law does not permit her to make her changes available to her students or to the public. (Some might argue that fair use allows her to make the changes themselves available to her students without redistributing the whole work, but that issue is beyond the scope of this paper.) Thus, it makes a practical difference whether or not AV developers think to post a license declaration along with their source code when they make it available.
3. Relevant FOSS infrastructure

Open source infrastructure can play two roles in the AV community. First, there are the potential advantages that come from making artifacts and systems open source. Second, developers of AVs might take advantage of the large number of tools and best practices available from the open source world to improve their software development efforts. Some of these tools are available for use no matter what license an artifact falls under, while others require that software derived from or connected to the tool be open source itself. This section describes tools that might be useful to an AV project. The next section describes the degree to which these tools appear to be used by the AV community (the short answer: hardly at all).

3.1. Java and HTML5

Since its appearance in 1995, Java overwhelmingly has become the platform of choice for AVs available on the web. In late 2006, Sun Microsystems announced that Java would be licensed under the GPL [16], thus bringing it into the true open source community. Because Java is licensed with the so-called “Classpath Exception” [17], AV projects are not required to be open source to use the newly-opened Java tools.

In the past year or two, HTML5 has matured into a robust platform that solves most compatibility issues that previously made Java attractive. HTML5 is supported by all major browsers, and software developed using it often runs on mobile devices and tablets as well, which is not necessarily true for Java. The two most recent AV development projects of which we are aware [18,11] both target HTML5. This might well be a trend for future AV projects. From a FOSS perspective, many JavaScript libraries and support tools (such as the popular jQuery library [19]) are open source.

3.2. GNU toolchain

The main open source compiler alternative to Java is the GNU compiler collection, especially the C++ compiler g++. Other tools included in the GNU development toolchain include the emacs editor, autoconf and automake for configuring the build process, and libtool for creating portable libraries. The GNU tools have a similar exception in their license that allows developers to use the tools on non-open source software (actually, non-Free software [20]) to be distributed under whatever license its creators choose.

While C++ is no longer popular for AV development, the other GNU tools can still be used even for web-based projects. The names and descriptions of the various components hint at the reason for adopting the GNU tools: portability. AV projects are most useful when they work on any platform, and cross-platform compatibility begins with portable code. Instead of targeting a different compiler on every platform, code that configures itself with autoconf can be written once and then run on any UNIX-like system (and sometimes Windows). Portable toolchains are beneficial for configuring complex client/server systems (such as JHAVÉ [21], TRAKLA2 [22] and OpenDSA [11]).

3.3. Version control

Another important tool for software development is source code version control. Here again, the open source community has much to add, in the form of the Concurrent Versions System (CVS), Subversion (SVN), and Git. Version control allows software developers to track changes to bodies of source code, connect those changes to individuals, and collaborate across physical boundaries (distance, time zones) using the internet.

Setting up a project to use any of the popular version control systems is fairly easy, allowing developers to quickly focus on building their visualization system rather than fighting with infrastructure. This small investment of time opens up development to other people at different institutions, thus potentially spreading the cost and effort across a larger group. Also, when a build of the code breaks, developers can go back to the last successful build and figure out what changes were checked in and by whom. Finally, source code repositories can be used either for open or non-open projects. All version control systems have built-in authentication controls to allow only certain individuals to commit changes, and either anonymous or authenticated browsing of the source code.

3.4. Issue tracking

Issue trackers allow project team members, and in many cases users or other community members, to report and track the status and progress on “issues”. An issue can be a bug report, a request for a new feature, an enhancement to an existing feature, a milestone toward which the developers are working, or any other project-related task project members wish to track. Issue trackers provide a form of institutional memory while also facilitating communication about issues, not only among team members, but also with potential external contributors or future researchers/developers. Further, issue trackers allow developers to ensure that community-generated issues come to some resolution, and aid in planning incremental updates and future releases.
3.5. FOSS hosting

For projects needing a permanent home on the internet, there are several widely-known choices including SourceForge [23], Google Code [24], and GitHub [25]. In addition to a web-accessible version control repository, these sites typically provide facilities such as web space, a compiler farm, mailing lists, bug tracking, and other infrastructure to aid in software development. Such sites nearly always require hosted software to be developed under an open source license.

Why would an academic developer of a visualization choose to host her project with an outside site, when almost every university provides web space for such research projects? SourceForge [26] provides several good reasons in its online documentation. Summarized, those reasons are:

• It simplifies maintenance,
• It helps meet license requirements,
• It promotes re-use,
• It allows continuation of orphaned projects, and
• It permits viable forks.

Probably the biggest reasons are easy access to version control, issue tracking support, and greater public visibility. If you want your project to be recognized as open source and you want it to be easy for others to find, SourceForge, Google Code, and GitHub all help with both goals.

4. FOSS infrastructure use

To explore how FOSS infrastructure is actually being used by the algorithm visualization community, we examined the sites by hand that were associated with the sorting visualizations in the AlgoViz catalog at the time [14]. We had to examine the associated sites since the catalog entries do not capture information about FOSS infrastructure use.

The results were extremely clear. Essentially no AV developers at the time indicated that they use open source tools or methods to create algorithm visualizations. Note that most of these projects were active in the early 2000s, and certain practices that are now common such as using open-source repositories for version control were possible but not widespread at the time. One AV project (out of about 50 projects surveyed) was hosted at FreshMeat (a mostly-open-source project hosting site like SourceForge). Another two indicated that g++ was used to build them. Aside from use of Java, these were the only instances of open source infrastructure use we were able to identify at the time, though we since learned from personal contact that at least one of the other projects used SVN internally, and it is likely others do make some internal use of FOSS tools as well.

Since then we have continued tracking the AV community through the AlgoViz catalog. Yet, virtually none of the new projects that have started in recent years (aside from a few associated with the AlgoViz site community effort described in Section 7) indicate availability through a repository site like GitHub or SourceForge.

5. Obstacles to FOSS in visualizations

Our survey shows that the AV community has not adopted many open source practices, nor do many AVs themselves meet the requirements to be considered open source. Why not? Some obstacles are inherent to any academic software development effort, while others seem specific to AVs.

5.1. Choosing a license

As the survey indicates, a significant fraction of AV projects that do not conform to open source standards fail to do so because they have no license. As we have not surveyed the authors, we cannot know for sure why so many lack a license. But it is likely that in most cases where an author posted AV source code without a license, it is simply because it never occurred to them that selecting a license was important. Another potential impediment to declaring a license is confusion over which license will best meet the needs of the authors, or the authors’ concerns over potential changes in their needs in the future. For example, one of this paper’s authors has been involved with a project that delayed declaring a license for months while trying to sort out the implications of a license choice. Questions arise such as “What is the difference between public domain and the BSD license? Which version of the GPL should our project choose? More importantly, why should I have to become an expert on software licenses? We already post our source on the web!” Atwood [27] provides a typical example of the public discussion and concern surrounding the issues with selecting a software license. The issues are no clearer since that article was posted in 2007.

We should note that the projects represented in the AlgoViz catalog span completion dates from the late 1990s to date. Over that time, open-source license types have become better known, along with the availability of open-source repositories (which at least implicitly push developers to declare a license by making this a requirement for using the site).
5.2. Institutional IP policy

Many universities and companies have strict policies about how institution-funded intellectual property can be distributed. For example, Virginia Tech’s IP policy [28] explicitly includes “software programs” and states that the copyright rests with the university, not the author(s) of the source code. Without holding copyright in the first place, an author may not release his or her work under an open source license (or indeed any license at all). However, another part of the policy states that “[t]he university’s mission includes dissemination of IPs in the most efficient and effective manner possible.” An author might argue that posting the AV’s source code with an open source license is “the most efficient and effective manner possible” for getting that visualization into the hands of educators. While University IP managers typically will agree with this view, it is still no surprise that some people do not want to wade into open source. Policies about ownership and distribution appear to them unclear and sometimes even contradictory!

5.3. Project management

Developers of projects like the Apache web server and Linux are well-versed in managing open source projects. They know about the various licenses, what each license obliges them to do, how to provide source code, and how to manage the potential flood of patches and new feature submissions. But what about the professor accustomed to having a few students informally create a set of AVs? The management concerns here are different because the scale of the effort is different. The professor might not have the time or inclination to deal with these issues. Academics also often are not experienced with such software development practices.

Some of the value provided by FOSS infrastructure sites and tools might (rightly or wrongly) be perceived by some AV developers as not applying to smaller projects. For example, consider the use of versioning systems. A professor might be used to “trusted” submissions from a small set of known individuals, where the form of these submissions can be clearly described and understood. A manager of an open source project, on the other hand, must treat every code submission as a potential Trojan horse containing malicious functionality. Worse, even if the manager can trust submissions, she must still deal with a variety of different mechanisms by which they might be contributed. Some people send a small patch file via email, while others prefer to receive commit access on the version control server. It would seem ungrateful to dictate one method or the other for receiving what is essentially a gift, so she does not require one approach.

Then there is the question of what the license requires. The GPL, for instance, requires the source to be distributed along with the binary. The project manager needs to ensure that the source to the latest release is available, and also must meet any other criteria set forth in the license. This can seem like an overwhelming task to someone who just took the plunge into open source.

Some development tools that are commonly available at sites like GitHub and SourceForge might appear irrelevant to a small project manager. Issue trackers are a typical example of such a tool. However, even for small projects, an issue tracker can be quite useful as a way to record corporate memory, communicate between team members, provide internal documentation, and structure the effort. Setting rules for repository branching and tagging, for assigning release numbers, and for coordinating work by separate developers are all additional mechanisms that can be quite useful in an open source project, but that might seem overwhelming to a small project manager who is just beginning to learn how to use a repository.

5.4. Community pressure and support

In the Linux community, almost everything is open source as a matter of course. The mindset of Linux developers is one of sharing and openness. Even if a project has no other compelling reason to be open source, this community standard tends to push developers towards it. This same community standard is not present in the AV community, on the other hand. With only 12% of the “market” open, and historically little interaction within developers in the AV community to begin with, there is no pressure from the community to open up other projects. Worse, there are no resources for making it easy for AV developers to learn open source practices, and historically there has been no organized AV development community or suitable web sites for disseminating this information.

Community pressure would push some project managers over the hurdle of learning about licensing, patch acceptance, and the other problems. Community support would provide them with the tools they need to jump those hurdles. It is a chicken-and-egg problem, though. Someone has to go through the process without the benefit of specific resources, and then they (or others) must create those resources and start applying pressure on a few others in order to kick-start the process.

The AlgoViz project has attempted in recent years to remedy this situation, and there has been increased communication among some current AV developers. But many projects are not in active development, and it is difficult to get developers to go back and bring them up to higher standards. We can hope that these organizational efforts will provide some pressure to new projects.

5.5. Costs vs. benefits

Finally, there is always the “what’s in it for me?” conundrum. Without community pressure or support, the tasks of learning how to manage an open source project, navigate institutional IP policy, and understand open source licensing have
not been undertaken voluntarily by most AV developers. Potentially, open sourcing could provide patches and new features. But most AV developers do not recognize or appreciate the potential. Most AV artifacts are small enough to be reasonably well-tested in-house, and bug reports will trickle in from outside users with or without the source code being available. Why, then, should an AV developer spend the time and effort required to become an open source AV developer? What rewards will that developer reap from such an investment of resources? Until these questions are answered satisfactorily, the pragmatic developer will simply continue on the path he already knows, and that path has not generally been open source. However, as we have seen this is often due to easily remedied mechanical issues rather than unwillingness.

6. Improving the community

Any tool that eases the overhead inherent in software development is bound to improve outcomes for an AV project. Adopting an existing development infrastructure allows educators to focus on the pedagogical and educational goals of their project. The obstacles described in the previous section are not insurmountable. This section will describe potential solutions for overcoming each one in turn, along with some larger issues not specifically addressed above.

6.1. Licensing and policy

Choosing an open source license and dealing with IP policy are considered together because the solution to the former may greatly ease the difficulty of the latter. The various FOSS licenses do provide a great deal of flexibility to software producers. Looking at the list provided by the Open Source Institute can be overwhelming. There are now many discussions and resources available on the Internet that discuss the pros and cons of the various licenses. For most projects, though, the only truly sticky decision would be between a BSD- or MIT-like license on the one hand, or a GPL-like license on the other. The latter requires parties who redistribute a modified binary to also redistribute the source code to any changes made, while a MIT-style license allows anyone to modify the source code and redistribute the binary without passing along those changes. In other words, a MIT-style license gives potential commercial developers more rights to use your software without explicit permission. The developers must decide which type of protection is more appropriate on a project-by-project basis.

6.2. Project management

A little education for developers would go a long way towards easing the difficulty of project management. There seem to be no sites that document the software engineering process for AVs, open source or otherwise. However, OpenAlgoViz does provide a repository for a number of AVs [29], and the AlgoViz portal does provide some limited information related to the AV development process along with its catalog and large bibliography of AV-related research papers.

6.3. Community pressure and support

Given the low quality of many of the earlier AV artifacts [7], the need for a more professional approach to developing these artifacts is apparent. While this is an independent problem from the question of open source, we believe that a healthy open source-based AV community would drive quality upward, as would adoption of open source practices like version control in a public repository. The mechanism would be two-pronged. On one hand, having the potential of support from outside developers will demand better software engineering from the start (and possibly provide some psychological pressure to create a higher-quality artifact). On the other, these outside developers will submit bug fixes and feature enhancements, leading to a higher-quality visualization.

6.4. Attrition

AV projects have historically been quite unstable in the long run with respect to accessibility. They often disappear or move around on the web, as is typical for web sites that are personally maintained by individuals. (For statistics on the rate of disappearance of algorithm visualizations, see [7].) Even though a university might be willing to host faculty or student projects, once the developers leave that university, their site might essentially be lost.

Using a hosting site such as GitHub or SourceForge from the start can have a major impact on the long-term sustainability of typical algorithm visualization projects. SourceForge has mechanisms in place for taking over a defunct project, and will not delete orphaned projects except under extremely rare circumstances [26].

6.5. Benefits to the community

Adopting FOSS best practices initially might seem to benefit the community more directly rather than the developer. To a developer, sharing source code might seem to be for the benefit of others, as is officially choosing a license. We hope that the advantages of exposing one’s code to public scrutiny (and potential public feedback in the form of bug fixes or suggestions for enhancements) will help to convince AV developers of value of an open source path and adoption of best
practices. There is also the fact that many FOSS tools attempt to push users toward open source. For example, many repository sites such as SourceForge and GitHub require users to make their projects open source as part of their terms of service.

More broadly, being open source seems to align well with the goals of educational software developers, even if in the past they have not always been conscious of the relationship. That is, if an AV developer has gone to the effort to make their project public, with a mechanism for distributing their software through a website, then surely they want people to use the software. At least implicitly, these developers are aiming for community impact. Becoming aware of, and becoming a part of, the community of AV developers and users should therefore be beneficial to those developers in that it should help to realize this goal of community impact. Further, awareness of the community can in turn have an impact on the AV developer, in that they can more easily learn about best practices and the existence of similar AV projects or improved pedagogical approaches.

7. OpenAlgoViz and OpenDSA

The original site that eventually became AlgoViz.org was created in 2006. Its initial mission was to provide a catalog of AVs to educators, and also to foster sharing of information within the AV community, including some of the software management issues described above. In 2009, we initiated the OpenAlgoViz repository at SourceForge. This is intended as a common location where various AV projects can house their source code in an accessible, open-source way. Since then, more than a half dozen development groups have contributed code to OpenAlgoViz. Generally, these were projects with pre-existing code bases that were moved to OpenAlgoViz as a way of banding together in a more organized way. In some cases, the move meant formally adopting a recognized open source license (as required under the SourceForge terms of use). For some—but not all—projects, this was the start of using version control to manage changes to source code. Some of the projects have simply housed the code base for an old project there, while others are still actively maintained.

In 2011, we initiated the OpenDSA project [30,9,10] including collaborators at Virginia Tech, Aalto University in Helsinki, and University of Wisconsin–Oshkosh. OpenDSA was conceived in part as a response to some of the impediments to AV adoption reported by instructors [2,4]. The OpenDSA Project seeks to provide complete instructional materials for a data structures and algorithms (DSA) course. OpenDSA’s goal is to build a complete open-source, online eTextbook for DSA courses, that integrates textbook quality text with algorithm visualizations and a rich collection of interactive exercises. All exercises are assessed automatically. As a result, students will gain far more practice by working on OpenDSA exercises than is possible with normal paper textbooks.

OpenDSA was designed based on what the AV community has learned from decades of research on developing AV systems and on developing pedagogically effective AVs. Some aspects are deliberately modeled on other successful open source projects, with aspirations of building community involvement in the design, implementation, and adoption of the project. For all of these reasons, OpenDSA can serve as a model for how to manage other open-source AV projects. OpenDSA is distributed using the MIT license. This means that anyone is free to reuse it, modify it, and redistribute it. OpenDSA is maintained through a series of GitHub repositories [31].

OpenDSA encourages reuse of content by course instructors. Instructors can modify the content for a module. (We use reStructuredText [32] for our authoring environment, so editing text is relatively easy.) Instructors can also pick-and-choose from among existing materials by means of a configuration file that can be used to generate a “book instance” with just the content that they want.

Support for building AVs is a core requirement for OpenDSA, since they are integral to the eTextbook concept. AVs are built on the JavaScript Algorithm Visualization (JSAV) library [33,34], itself an open source project. Building an AV support system and constructing a large body of AV content is a huge undertaking. Thus, we need to recruit as large a community of developers as possible to create content. Maintaining the body of content is a huge undertaking as well. Fortunately, many of the components are independent of each other; for example, the individual AV implementations are independent from each other. Thus, it is reasonable to hope that individuals who are not intimately involved in the core of the project can still provide bug fixes or enhancements for various components.

Support for a wide and flexible array of interactive exercises is another core requirement. We want to support everything from static multiple-choice questions to dynamically-generated random problem instances for interactive exercises with non-trivial user interfaces. These exercises need to be automatically assessed by the system. This has led us to adopting the Khan Academy framework [35], also an open source project, for underlying support for certain types of exercises. We hope that we can become successful beneficiaries of a community of exercise contributors in the way that Khan Academy has.

Relatively few AVs have been written in JavaScript, while the Java infrastructure necessary to run nearly all existing AVs is becoming increasingly difficult to sustain. Thus, there will be much incentive in the near future to write new AVs that target HTML5. We hope that the existence of OpenDSA will help to guide developers of new AVs in a positive way. Whether they use portions of our code base, such as the JSAV library, or whether they just follow our model for good open source practices, hopefully the result will be a steady improvement in the AVs developed.

Initial student reaction to OpenDSA in the classroom has been overwhelmingly positive. OpenDSA was used in a data structures class at Virginia Tech for a three-week period during Fall 2012 to teach the topics of sorting and hashing. Prior to its use, the students were surveyed regarding their willingness to use an online interactive textbook, with the mean score being 3.88 on a 5-point Likert scale. After their experience with OpenDSA, they were asked a roughly equivalent question.
about how much they liked using an online interactive textbook. Their opinion rose to 4.30 on the 5-point scale. Compared to a control group, OpenDSA users’ test scores on this material were higher, but not by a statistically significant amount. During Spring 2013, students at three Universities used OpenDSA material, and expressed similarly positive opinions on the experience. Further details on our experience with using OpenDSA in the classroom can be found in [30].

8. Conclusion

Along with evaluation of pedagogical effectiveness, there has also been some study of the adoption of AVs into CS courses [36–38,4]. Surveys [36,4] of instructors have shown that the penetration is not so high as advocates would wish. Instructors indicate that finding high-quality software is an issue. Even when such high-quality software exists, then it is reasonable to expect that the confusion generated by a large body of low-quality software without adequate guidance from a reputable source will have a negative impact on adoption. To the extent that improved development practices lead to better software products, the educational community will benefit by their adoption.

So why should the AV community care about open source? There are two classes of answers, the pragmatic and the philosophical. Pragmatically, the tools available at open source sites like GitHub and SourceForge are meant to improve productivity. The open source approach provides for better user feedback and improvements to the software. Educators may have limited resources in terms of developers and testers for their software, and duplication of effort across institutions is inefficient. Sharing an open source project allows the easy combination of these and other resources.

The Open Source Initiative [39] summarizes the open source message in this way:

When programmers can read, redistribute, and modify the source code for a piece of software, the software evolves. People improve it, people adapt it, people fix bugs. And this can happen at a speed that, if one is used to the slow pace of conventional software development, seems astonishing.

Philosophically, the idea of collaboration is already a part of the research culture in higher education. Open source development extends this culture of collaboration to the education side of the mission. The open source community itself provides educational benefits to the students who often are doing the software development on AV projects. It helps them learn to work in a distributed team, building a “real” code base [40].

References


[34] JSAV, JSAV project repository, 2013.