Problems with using components in educational software
Anne Morgan Spalter*, Andries van Dam

*Corresponding author.
E-mail addresses: ams@cs.brown.edu (A. Morgan Spalter), avd@cs.brown.edu (A. van Dam).

Abstract

Reuse is vital in the education world because the time and money necessary to create high-quality educational software is a significant problem. Estimates for the cost of creating a single well-designed, highly graphical and interactive online course in the commercial domain range from several hundred thousand dollars to a million or more. Thus, the idea of reusable software components that can be shared easily is tremendously appealing. In fact, “component” has become a buzzword in the educational software community, with millions of dollars from the National Science Foundation and other sponsors funding a wide variety of “component-based” projects. But few, if any, of these projects have approached the grand vision of creating repositories of easy-to-reuse components for developers and educators. This paper investigates some of the factors that stand in the way of achieving this goal. It also looks forward to a new genre of educational software that we hope will emerge when the basic components problems have been addressed.

We begin by defining the word “component” and looking at several projects using components, with a focus on our Exploratories project at Brown University. We then discuss challenges in: critical mass, intellectual property issues, platform and system specificity, programming in the university environment, quality assurance, searching and metadata, and social issues. We look at relevant software engineering issues and describe why we believe educational applications have unique factors that should be considered when using components.

Keywords: Components; Education; Educational software; Reuse

1. Why are components of such concern to the educational community?

The concept of reusability for sharing elements of educational software is of special concern for the education community. Reuse is vital in the education world because the time and money necessary to create high-quality educational software are major inhibitors. Estimates for the cost of creating a single well-designed, highly graphical and interactive online course in the commercial domain range from several hundred thousand dollars to a million or more. Thus the idea of reusable components that can be shared easily is tremendously appealing, and few argue that reuse should not be fostered at all levels.

Reusable components should be of particular interest to the computer graphics education community because some of the most frequent types of educational components are graphical elements, from user interface widgets to visualizations and simulations. In the course of creating several dozen educational Java applets for the field of computer graphics, we have identified many areas that would benefit greatly from the existence of component building blocks, especially in the 3D domain where writing from scratch is especially difficult and time-consuming. Our future work, described later in this paper, builds on this observation.

2. What, exactly, is a component?

Although there is virtually complete agreement about the desirability of reusability, the word “component” is subject to many different interpretations. The most
frequent interpretation in the educational community seems to refer to any type of material that can be reused in an educational effort, from a lesson plan to an applet to an image, a video clip, or a piece of code. This meaning is in keeping with the dictionary definition of a component as simply “an ingredient” and also seems synonymous with the vague term “learning object,” which is used to identify any part of a learning project or “environment.”

This all-encompassing definition of “component” obscures many issues, however. The problem of keeping track of a large collection of applets or media elements such as film clips is, on the technical side, chiefly a database and user interface problem. The problem of creating pieces of code that can be plugged together by programmers, however, is another sort of technical challenge that concerns a different audience, one of software developers. We are interested in the latter meaning and, in order to distinguish between our usage and the more general sense of the word, we will refer to reusable pieces of code as “software components.”

Like physical construction kits, from Erector sets to Lego, software components should let their users build a multitude of experiences from a single set of tools. Although connecting pieces of code is much more complex than connecting pieces of a construction set, the analogy can provide a useful mental model for non-programmers.

All software components have certain high-level characteristics in common: they are pieces of code that provide documented, standardized means to access the functionality they contain. Popular standards include, among others, Microsoft’s component object model (COM), Microsoft’s .NET assemblies (component class files stored with associated metadata), Sun’s JavaBeans or Enterprise JavaBeans, and Borland’s Delphi VCLs.

Within this high-level definition, software components vary by how much a user can alter them. At one end of this spectrum lie “black-box” components that are typically distributed in compiled or binary form and cannot be changed easily, if at all. Programmers have access to documentation about a black-box component’s functionality and how that functionality can be accessed, but cannot change its code. At the other end of the spectrum lie “white-box” components, which are typically distributed as source code, and are therefore open to full inspection and change by programmers. Which end of the spectrum one favors is a matter of passionate debate among software engineers. Obviously, there are trade-offs with each approach: a black-box component may be easier to use and test but may lack a key piece of functionality that the programmer desires. The white-box approach lets programmers modify components easily but can easily lead to maintenance nightmares and make quality assurance more difficult.

Our Exploratories project at Brown University [1] creates Web-based learning materials for the teaching of introductory computer graphics (currently using Java and Java3D). For this project, we have adopted a “gray-box” approach in which we make the internal structure of complex components visible to programmers and permit them to replace small “black-box” pieces or rewrite them as needed. This approach minimizes unforeseen internal dependencies that can occur in black-box component assembly while also avoiding some of the maintenance complications of full-fledged white-box components. Fig. 1 shows some of the different levels of component granularity in our repository. All of our component-based Java applets use the moderately large-grained Exploratory component, which handles all basic windowing and menus. Most of our 3D applets use the 3D axes component, but programmers can always access internal elements, such as the axes’ line parameters, by changing the line component at the bottom of the axes hierarchy. More detail on our approach to component granularity can be found in [2]. We recommend Clemens Szyperski’s book Component Software [3] for more detail on technical and social issues of software component use.

3. What educational software projects are developing components?

A number of educational projects have been tackling the problem of creating reusable software components for educational applications, including our Exploratories project, which publishes findings about strategies that seem to work for creating and using educational applets [2,4,5]. In addition to our own experiences, we interviewed Chris DiGiano, a leader of the ESCOT project [20] that created component-based software for middle school math. We also drew on e-mail correspondence with Manolis Koutlis of the E-Slate project [6], a large-scale, long-term (over eight years now) undertaking in Greece to create an easy-to-use visual component assembly environment for teachers. Our understanding of both ESCOT and E-Slate were assisted immeasurably by reports from Andy diSessa’s Web/Comp project [7], which explores issues in component-based educational computing. (Please note that the definition of component computing used in the Web/comp project is more inclusive than the one used in this paper.) DiSessa’s write-up of his own project, Boxer [7], also sheds light on this subject. Finally, the Brown Exploratories project is part of the NSF National Science Digital Library (NSDL) program [8] and through annual meetings with fellow grantees we have become familiar with many issues in organizing repositories of educational materials.
We focus on the few projects mentioned above because these are the only large-scale educational software efforts we could discover that are attempting to produce sustainable libraries of educational software components. While there are many projects that assemble collections of various types of stand-alone educational materials, such as applets (the Educational Object Economy discussed by DiSessa is a good example, as is Europe’s large Ariadne Foundation’s collection [9]), there seem to be very few that are building repositories of component-based building blocks in the sense in which we define them in the previous section.

Despite some successes in these projects and others, and despite significant contributions from a range of talented and hardworking people, educators and programmers working with educators cannot yet go to the Web and search repositories of freely available reusable software components. We asked ourselves why all the work on components has not resulted in more of them and what can be done to fix the situation.

It is important to note that several projects, including E-Slate, have also been working on the development of easy-to-use, predominantly visual environments that will allow non-programming educators to combine components on their own. E-Slate is currently offering free
downloads of both its runtime and development environments [6]. Boxer [7], which lets educators create software in a custom environment that has many component-based aspects, also offers free downloads. Our collaborators for the NSF NSDL project have developed an environment called CreateStudio that builds on instructors’ familiarity with simple Web design and allows them to create links between customizable interactive components. CreateStudio is being tested and used in a chemistry course at Carnegie Mellon. Information on CreateStudio and free downloads can be found on the group’s Web page [10]. These and other assembly tools being developed in the NSF NSDL should help bring teachers closer to the creative act of assembling their own educational software. Developing such assembly tools is a research problem in and of itself, and we do not investigate it further in this paper.

4. The problems with components

This section contains an alphabetical listing of central issues for component-based educational software projects and describes some of the problems we and others have faced or anticipate facing in the near future.

4.1. Critical mass

The leverage provided by components cannot be realized without a critical mass of components. But the motivation to build component-based systems may be lacking if that critical mass does not already exist. This chicken-and-egg problem must be addressed for the vision of truly useful (i.e., high-quality and well-populated) repositories of educational components to become a reality.

There is ongoing debate about the number of components needed to establish a useful repository in a specific domain and about the level of granularity that is most effective. ESCOT and E-Slate both favor the inclusion of large-grained, somewhat modifiable components (such as the Geometer Sketchpad component in ESCOT or E-Slate’s Map, Database, or Grapher components). Exploratories, on the other hand, has focused on making all aspects of even large-grained components highly accessible to programmers, resulting in hierarchies of many fine-grained components (see Fig. 1). It may be that a repository designed for use by teachers who are combining pre-made components should contain a much smaller number of components (and chiefly larger-grained ones) than a repository designed for programmers who want to be able to make modifications at all levels (from what a simulation does down to its look and feel). The E-Slate repository listing, shown at the top of Fig. 2, presents chiefly large-grained components such as a full-fledged database program and a logo-like scripting environment (although some finer-grained ones, such as a vector component, are offered too). The bottom part of the figure shows the Image Editor component, which provides a tool for general bitmap editing and freehand drawing.

4.2. Intellectual property (IP) issues

The time and money necessary to create a working set of reusable components are most often prohibitive for not-for-profit ventures. Thus for educational institutions, it can make sense to partner with commercial ventures or to try to market some aspects of their work independently. The ESCOT project partnered with several companies that contributed greatly to the component content, particularly simulations. One of their main partners, AgentSheets, helped them generate simulations quite rapidly. Under the terms of the grant, many components were used for free for experimental educational work. Now that the grant is over, however, it leaves behind no free repository for others to use. Many IP issues have kept ESCOT from more broadly releasing their full set of components. For example, one of the most frequently used components in the ESCOT project uses the commercial product Geometer’s Sketchpad, from Key Curriculum Press, to generate components consisting of geometric constructions and animations. The Sketchpad components can then be
triggered by other ESCOT components. Fig. 3 shows the Geometer’s Sketchpad interface with examples of components generated for different ESCOT-created math Problems of the Week (PoW), as seen on the Drexel MathForum [11].

IP and licensing are standard issues in the corporate world and corporations have legal counsel and often their own team of lawyers. While university faculty commonly have some level of free legal counsel available for issues related to their work, teachers at K-12 schools are not accustomed to evaluating licensing options for code or paying for pieces of code and are likely to avoid anything that is not offered for free.

4.3. Platform and system specificity

Many projects, ours included, have run into enormously time-consuming problems created by platform specificity. This problem is not unique to education but is especially important because the types of computer platforms used in schools vary widely, and many are relatively old, compared with corporate systems. We have given up trying to make our applets work on the Apple Macintosh series, for instance, because the Java plugins are routinely released months or even years behind those for the PC. Microsoft’s .NET technology looks interesting, but does not work under Unix, which runs on the bulk of the machines in our research lab.

We have frequently encountered problems with different types of browsers (an applet works with one and not another) and among different versions of the same browser. In addition, the setup of the system software and networking for the machine (with attendant permissions issues) can cause problems. Both E-Slate and ESCOT have run into the same or similar issues in trying to roll out content into schools [7]. E-Slate now uses its own software for both building and using its educational environments. Although these problems may seem trivial, they consume a disproportionate amount of time and can dramatically erode user confidence.

A partial solution to this problem, and one that has been adopted by our collaborators at Carnegie Mellon, is to package the software as well as the environment required to run it on a CD-ROM and distribute this physically to each student using the software. This lessens software incompatibility issues but does not resolve inter-platform or hardware compatibility problems. Such a solution also significantly lessens the impact of electronic learning since the course materials can then no longer be distributed electronically.

4.4. Programming in the university environment

Unlike the full-time programmers in a corporation, the programming staff of a university educational software project is often composed of undergraduates or master’s students. Although this approach provides a rich educational experience for the students (and can also result in wonderfully creative ideas), we have found...
that the learning process for component creation is much more difficult than we had imagined. Even after indoctrination into object-oriented programming and the benefits of reusability, and at least a month of motivating and then heavily using components (JavaBeans), we found a wide range of real understanding of what a component is and how one should be designed for reusability. Students were also unclear about how their components might interact with those from other sources. The E-Slate project has found that it takes about two years for an experienced developer to design truly reusable components.

In addition, even when students exhibit a reasonably strong theoretical understanding of how to design, write, and reuse components, and are able adequately to convey these ideas verbally and in writing, they are often unable to translate that knowledge into practical skills. Their software therefore seldom lives up to their own expectations or those of their collaborators.

Component projects undertaken in an educational setting not only may have to cope with inexperienced programmers, but are usually highly constrained by funding logistics. A grant may run for a period of only two, maybe three, years (E-Slate has been developed over the course of nine separate grants). Hiring talented staff without being able to offer any sense of job security can be difficult.

4.5. Quality assurance

How can programmers tell if the components they are downloading will work as promised? Will component source vendors fully test the wares they are selling? Few, if any, academic educational software projects employ any systematic testing of component compatibilities. Even testing by commercial component vendors such as ComponentSource [12], a commercial Web-based marketplace for components, is composed only of testing for viruses, completeness, installation and de-installation. The projects discussed here are all in the early stages of creating full-fledged component repositories, and the original developers of the components are usually on hand to provide technical support and make changes. In the near future, however, resources will have to be directed toward testing efforts if the repositories of educational components are to be trusted (a feature that is essential to make the whole component model work).

Another approach to quality assurance lies in Amazon.com-style user reviews. ComponentSource offers user review sections and discussion groups for each of its components. The Merlot repository of educational materials employs a peer review system (using teams of handpicked specialists) as well as user reviews for its content; their framework could also be applied to software components [21].

4.6. Searching and metadata

The repositories created as part of ESCOT’s and E-Slate’s work have so few entries that locating items is not yet a serious issue (each has approximately 30 components). Components can be organized by categories that appeal to each project’s programmers or organized by the demands of different projects. For example, AgentSheets, a company that worked with the ESCOT project, provides an environment in which non-programmers can create graphical “characters” with behaviors. These characters can be exported as JavaBeans. Over 500 simple AgentSheets components are available on the AgentSheets website [13], organized by the name of the projects for which they were created.

While these strategies work with small numbers of components, different strategies are necessary for larger repositories. The two main approaches used in other types of reusable asset repositories will probably also work with repositories of software components: organization by topic and searching through metadata. A great deal of work on the manual and automatic generation of metadata has been done as part of the NSF Digital Library initiative. In particular, the NSDL has addressed issues specific to educational use of metadata and the environments in which it is used. Research has also shown that for domain-specific repositories of limited size, keyword searches without use of any metadata can be quite effective [14].

4.7. Social issues

Finally, significant barriers exist in the sociological aspects of educational software component development. The ESCOT project often found that social and organizational issues had more impact on the results than any of the technical issues. The mentoring of both teachers (in the design and use of the software projects) and the software developers (in ways of working collaboratively with the teachers) often seemed to be a more important factor than any advances in the ease of coding.

Some of the problems crucial to understanding the use of components in K-12 education come from the need to combine strong software engineering skills, deep domain knowledge, and pedagogical/instructional design knowledge. The chief holders of these types of knowledge tend to have different backgrounds, personal goals, and terminologies. While our project at Brown largely circumvents such issues by having educators who not only talk the language of programming but actually teach it, the ESCOT project found first-hand that mixing these cultures can be a challenge. ESCOT used “integration teams,” pairing programmers and designers with educators and educational technologists to design software that would have immediate classroom use.
Although they confirm that having the classroom teachers in the design loop from the very beginning is essential, they acknowledge that communication can be difficult amongst the different groups.

5. Has anyone succeeded with reusable components in any domain?

Although the educational community has not yet had remarkable success stories with components, the picture is brighter in other fields. Large numbers of Microsoft VisualBasic components are available online. Marketplaces such as ComponentSource and Flashline have thousands of offerings and claim hundreds of thousands of downloads. Companies such as BEA [15], which sells component-based e-commerce software, provide proof of workable business models for the use of components in real-world situations.

Design and implementation strategies for creating component software have been developed at institutions such as Carnegie Mellon’s Software Engineering Institute (SEI) [16]. The SEI group advocates a “product line” approach, in which reusable components are developed for specific domains according to strict software engineering guidelines. Such an approach can ensure that components offer robust functionality but are not so complex or overloaded with such a wide range of functionality that programmers simply write new, simpler functionality for themselves. A coherent collection of components is much more likely to be worth learning how to use than a random collection.

Component-based reuse has also been judged a success in military projects and government institutions such as NASA [17]. Although some fundamental research problems remain for general cases of component reuse and composition, many challenges have been met, including methods of domain analysis and ways of measuring the benefits of reuse [14].

Although there certainly have been successes, the SEI documents and virtually every other strategy or case-study paper reconfirm our own experiences that writing reusable code requires tremendous overhead. It may take many times as much effort to create a set of reusable components as to develop code for one-time use. In many cases, it does not make sense to stress reuse. We have found that the final design of a reusable component must be completely solidified before undertaking this decidedly more demanding approach to programming.

6. So is this just a software engineering problem?

It is natural to ask why, if components are working in one domain, they aren’t working in another. Is it simply that the proper software engineering knowledge has not been applied to educational uses or that the solutions to relevant problems are not even known? Our feeling is that the answer is no: that the lack of success in creation of reusable component building blocks for education cannot be ascribed solely to software engineering issues. Although better software engineering practices are factors and almost certainly are necessary, we do not think such knowledge and application of certain practices are, by themselves, sufficient. Also, although the component problem is hard in any field, we do not believe that current limits of research knowledge are what is holding back educational efforts.

Our own experience and our discussions with others, in particular Chris DiGiano of the ESCOT project and Andy diSessa, reveal that production by and for the educational community differs substantially from that in more commercial settings. Social issues, platform problems, and severely limited funding and timescales preclude the application of many software engineering recommendations, including, say, establishment of reuse teams in addition to regular programmers [16].

7. Looking forward—large-scale use of next-generation educational components

Once we successfully address the barriers discussed in Section 4, we will be able to focus on component repositories that leverage both the increasing availability of 3D graphics and a better understanding of how to model and simulate complex systems. While today’s educational component-based efforts are usually targeted to specific age-levels, grades, and levels of intellectual sophistication (such as students in an introductory physics course in a community college), such an approach results in numerous sets of materials and no obvious way to relate or combine them. For example, the human heart may be modeled as a simple biological pump for a middle school science course and as a vastly more complex entity for a medical student.

To fully leverage the component advantage, we anticipate a need for interoperability between these many levels of component complexity and sophistication. One can easily imagine, for example, that a student studying cell physiology at the level of the Krebs cycle may need a quite detailed cell model that can interoperate with far more superficial heart, lung, vascular system and kidney models. To address this need, we have been developing the concept of families of “clip models” to indicate that a component consisting of a 3D simulation model should be designed, from the beginning, as an entire family of such components. Each member of the clip model’s family would support a different level of sophistication, complexity, and perhaps
even learning style. The challenge lies not only in developing strategies for making such families of clip models, but in making them all interoperable at all their different levels, as in the model interoperation that would be required for the combination of the “detailed cell” and “simple heart” models in the example above [18].

Although we have addressed some of the component interoperability issues in our Exploratories project, the combinatorial explosion presented by interoperating clip models at multiple levels is a challenge an order of magnitude more complex than current reusable software component work. This challenge has not, as far as we know, been targeted, let alone solved, by the software engineering community.

In addition to the software engineering and pedagogical issues, there are also serious ontological issues involved. When you have one or at most a small team of authors writing a single book targeted at a single audience, the domain specification as seen in the definition and relationships of concepts and terms is a challenging but manageable task. When you expand the context as described above, the situation suddenly becomes more intractable. The Knowledge Web community is now starting to tackle the problem of identifying and encoding domain-specific ontologies for the Web. Holsapple and Joshi [19] describe a collaborative approach to designing an ontology that begins with independent ontological proposals from several authors and incorporates input from many contributors. Some sort of collaborative approach to ontological engineering will have to be developed in order to build an ontology that is acceptable to many members of a given field. We believe that an effective ontology for the clip models described here requires collaboration among computer scientists, subject domain experts, and educational experts.

8. Conclusion

A fully generalized vision of components, in which one can go to a Web site and grab many pieces of code written by different people for different domains and easily assemble them into an application, may never come to pass. This vision may, in fact, be technically impossible. But a more limited goal, with domain-specific sets of components written to standard specs and tested before being released, still has the power to revolutionize educational software development, not only making it dramatically faster for programmers to create new applications but also making development accessible to non-programming educators through visual assembly environments. It is clear, however, that the level of awareness of components needs to be vastly improved, both to motivate production and usage and to help both project teams and funding agencies gain a realistic understanding of the needed resources. In addition, the vocabulary of components needs to be standardized so that we are all discussing the same things.

The full benefits of the type of component work going on today, and the type that may be underway in the near future, such as clip models, cannot be realized without an essential critical mass of components that large communities can use. We and several of our colleagues have proposed a Learning Federation, a non-profit consortium of government and industry, to provide long-term funding for the creation of novel tools and exemplary online courses using highly interactive computer-based environments. Some large-scale undertaking of this nature, with a focus on software components, may well need to be created before the full potential of component repositories for educational software can be realized.

Acknowledgements

This paper would not have been possible without the work done by Jean Laleuf as Technical Director of the Exploratories project. We are indebted to Shriram Krishnamurthi of Brown University for his technical insights and to Chris DiGiano of the ESCOT project for phone and email conversations about components. We also wish to thank Manoulis Koutlis of E-Slate and Andy DiSessa and his Web/comp project. Thanks also to Rosemary Michelle Simpson for her critical reading and assistance with the references.

Our Exploratories work has recently been sponsored by the NSF Science and Technology Center for Computer Graphics and Scientific Visualization, the NSF NSDL grant “A Component Repository and Environment for Assembly of Teaching Environments (CREATE),” and Sun Microsystems, Inc.

References

[1] Exploratories project. Thanks also to http://www.cs.brown.edu/exploratories/


