Problems with Using Components in Educational Software

Anne Morgan Spalter
Brown University
ams@cs.brown.edu

Abstract

Reuse is vital in the education world because the time and money necessary to create high quality educational software is prohibitive. 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 easily shared 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.

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: Searching and Metadata, Quality Assurance, Programming in the University Environment, Platform and System Specificity, Social Issues, Intellectual Property Issues, and Critical Mass. 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 is prohibitive. 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 easily shared is tremendously appealing and few argue that reuse should be not be fostered at all levels.

Reusable components should be of particular interest to the computer graphics education community because one of the most frequent (if not 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 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.

2 What, Exactly, is a Component?

Although there is virtually complete agreement about the end vision 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, 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. 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 these reusable pieces of code as "software components."

All software components have certain high-level characteristics in common: they are pieces of code that provide documented, standardized means by which to access the functionality they contain. Popular standards include, among others, Microsoft's Component Object Model (COM), 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 easily be changed. 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.

In our Exploratories project we have adopted a "gray box" approach in which we make the internal structure of more 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. More detail on our approach to component granularity can be found in E-slate [E-slate]. We recommend Clemens Szyperski's Component Software book [Szyperski 1999]

for a more detail on technical and related 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 at Brown University [Exploratories Project]. Exploratories creates Web-based learning materials for the teaching of introductory computer graphics (currently using Java and Java3D) and publishes findings about strategies that seem to work for creating and using this content [Laleuf and Spalter 2001; Spalter et al 2000; Spalter and Simpson 2000]. In addition to our own experiences, we interviewed Chris DiGiano, a leader of the ESCOT project [ESCOT]. ESCOT created component-based software for middle school math. We also drew on email correspondence with Manolis Koutlis of the E-Slate project, 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 [WEB/comp Project], which is funded to explore 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 [WEB/comp Project] sheds light on this subject. Finally, the Brown Exploratories project is part of the NSF National Science Digital Library (NSDL) program [NSF NSDL] and through annual meetings with fellow grantees we have become familiar with many issues of 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 various types of stand-alone educational materials, such as applets (the Educational Object Economy, discussed by DiSessa is a good example), there seem to be very few that are building repositories of component-based building blocks in the sense that 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 in 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, as well as Exploratories (through collaboration with Dave Yaron's Irydium group [CREATE] at Carnegie Mellon) 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 [E-slate]. Boxer [WEB/comp Project], which lets educators create software in a custom environment that has many component-based aspects, also offers free downloads. The Carnegie Mellon tool is still under construction. 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 [NSF NSDL]. 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 others and we have faced or anticipate facing in the near future.

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 the level of granularity that is most effective. ESCOT and E-Slate both favor the inclusion of large-grained, somewhat modifiable components (such as those created for ESCOT in Geometer's Sketchpad 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. 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).

Intellectual Property (IP) Issues: The time and money necessary to create a working set of reusable components is 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 one's 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 no free repository behind for others to use. Many IP issues have kept ESCOT from more broadly releasing their full set of components.

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 most likely to avoid anything that is not offered for free.

Platform and System Specificity: Many projects, ours included, have run into enormously time-consuming problems addressing issues of 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 won't run on Unix machines, which make up the bulk of the machines in our 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 [Agresti and Evanco]. 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 physically distribute this 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.

Programming in the University Environment: Unlike the full-time programmers in a corporation, the programming staff of an educational software project is often composed of undergraduate or graduate students. Although this approach is a rich educational experience for the students (and can also result in wonderfully creative ideas), we have found that the learning curve 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 was 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 to adequately 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, aside from coping with inexperienced programmers, are usually highly constrained by funding logistics as well. 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 offering any sense of any job security can be difficult. In addition, it can often take many months for a new hire to come up to speed.

Quality Assurance: How do programmers know 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 [ComponentSource], 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 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 (an essential feature for the whole component model to 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 employs a peer review system (using teams of handpicked specialists) as well as user reviews for its assets; their framework could also be applied to software components.

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 functional demands, such as 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 Java-Beans. Over 500 simple AgentSheets components are available on the AgentSheets web site [AgentSheets], 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 education sub-program of this grant, the NSDL [Laleuf and Spalter 2001] has addressed issues particular 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 [Poulin 1999].

Several groups now use metadata to harvest content and to make it searchable. The Merlot project [Merlot], for instance, is a continually growing collection of online learning materials, peer and user reviews, and assignments. Such projects could extend their systems to include software components and we are currently in discussion with the Merlot group about this potential.

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 or 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 was essential, they acknowledge that communication can be difficult amongst the members of such 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 Visual Basic components are available online. Marketplaces such as ComponentSource and Flashline [ComponentSource] have thousands of offerings and claim hundreds of thousands of downloads. Companies such as BEA [BEA], 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) [SEI]. 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 as 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 [Agresti and Evanco 1992]. 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 [Poulin 1999].

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 than simply 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 for the lack of success in the 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 to lift the educational component vision off the ground. Also, although the component problem is hard in any field, we do not believe that current limits of research knowledge are what are 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. The social issues, platform problems, and severely limited funding and time-scales preclude the application of many software engineering recommendations, including, say, establishment of reuse teams in addition to regular programmers [Poulin 1999].

To add to these difficulties, there are a great many complex and poorly understood issues that affect educational software, making it difficult to analyze in terms of components. These issues include topics such as pedagogy, assessment, and classroom implementation. Even if a set of successful components is created for a given domain, the problems of interoperating with components from other subject areas, or within the same subject area but at different levels of sophistication and abstraction, are formidable.

7   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 it into a 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 specifications 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.

Even with increased knowledge and better software engineering, however, we do not believe that the essential critical mass of components can be created without some coordinated effort to "prime the pump." Although creating a "product line" of software components for all of computer graphics or some other large field may be impossible for many years, a coordinated effort to address, say, all the linear algebra needed for introductory computer graphics, or a color theory unit, could set an extremely useful example. However, such a collection, from user interface elements to underlying math engines or simulation pieces, cannot grow ad hoc; it would require a sizeable up-front planning and design effort.

Andries van Dam and others have proposed a Learning Federation, a non-profit consortium of government and industry, to provide an usually long-term and high level of funding for the creation of exemplary online courses using highly interactive computer-based environments. Something of this nature, with a focus on software components, may well need to be created before the potential of component repositories for educational software can be realized.

8   Acknowledgements

References

AGENTSHEETS. http://www.agentsheets.com/.

AGRESTI, W. AND EVANCO, W.1992. Projecting Software Defects. In Analyzing Ada Designs in *IEEE Transactions on Software Engineering*, 18 [11], pp. 988-997.

BEA SYSTEMS SUCCESS STORY. http://www.componentsource. com/SellComponents/SuccessStories/BEASystems.asp.

COMPONENTSOURCE. http://www.componentsource.com/ and http://www.componentsource.com/Welcome/NewVisitor/ WhyBuyFromComponentSource.asp for their testing statement.

CREATE (A COMPONENT REPOSITORY AND ENVIRONMENT FOR ASSEMBLY OF TEACHING ENVIRONMENTS) PROJECT. http://www. smete.org/nsdl/projects/services.html#brwn

E-SLATE. http://e-slate.cti.gr/.

ESCOT (EDUCATIONAL SOFTWARE COMPONENTS OF TOMORROW). http://www.escot.org/.

EXPLORATORIES PROJECT. http://www.cs.brown.edu/exploratories

LALEUF, J. R. AND SPALTER, A. M. 2001. A Component Repository for Learning Objects: A Progress Report. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries*.

MERLOT. http://www.merlot.org/

NSF NSDL (NATIONAL SCIENCE DIGITAL LIBRARY) PROGRAM. http://www.smete.org/nsdl/projects/index.html.

POULIN, J. 1999. Been There, Done That. In *CACM (Communications of the ACM)*, 42 [5].

SEI (SOFTWARE ENGINEERING INSTITUTE), Carnegie Mellon. A Framework for Software Product Line Practice - Version 3.0, http://www.sei.cmu.edu/plp/framework.html

SPALTER, A. M., LEGRAND, M., TAICHI, S., AND SIMPSON, R. M. 2000. Considering a Full Range of Teaching Techniques for Use in Interactive Educational Software: A Practical Guide and Brainstorming Session. In *Proceedings of IEEE Frontiers in Education '00*.

SPALTER, A. M., AND SIMPSON, R. M. 2000. Integrating Interactive Computer-Based Learning Experiences Into Established Curricula. In *Proceedings of ACM ITiCSE (Innovation and Technology in Computer Science Education) 2000*.

SZYPERSKI, C. 1999. *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley, New York.

VAN DAM, A. 2000. Reflections on Next-Generation Educational Software. In *Enseigner L'Informatique: Melanges en Hommage 'a Bernard Levrat.* Pellegrini, C. and Jacquesson, A. editors, Georg Editeur, pp. 153-166. http://www.cs.brown.edu/people/avd/LevratPaper.html.

WEB/COMP PROJECT. diSessa, A. A. http://dewey.soe.berkeley. edu/boxer/webcomp/index.html.