OPEN SOURCE EDUCATIONAL APPLICATIONS

Bob Tinker, President
The Concord Consortium
http://concord.org

INTRODUCTION

Education is failing to exploit the power of modern personal computers. Although many sophisticated educational models and tools can be imagined that could significantly advance education, there are too few applications that take full advantage of the supercomputers now on students' desks and in their backpacks. The problem is that sophisticated, classroom-ready educational applications are beyond the resources currently available from grants or investors to develop and maintain. The solution is to change the paradigm used for software development to open source.

This paper describes how educators and programmers worldwide could create a shared collection of educational applications based on cutting-edge open source educational software that could help realize the potential of technology to transform education. A body of educational applications that are all free, open source, and maintained by a collaborating community, could result in an exciting new generation of computer-based learning activities that are well designed, robust, and highly effective.

The history of GNU/Linux is a spectacular demonstration that an open source strategy can result in superior code that is free and can still be the basis for commercial services and products. This paper is, however, not *about* Linux and does not take a position on the role of the Linux operating system in education. This paper is focused on open source educational *applications* that execute in any operating system, including Windows, Apple OS, *and* Linux.

There are many ways of using computers in education. One of the most promising directions is based on powerful tools and models that could be used in many different learning contexts (Feurzeig & Roberts, 1999; Gobert & Buckley, 2000; Tinker, 1990a, 1990b). Students learn from these tools and models through guided explorations of appropriate problems and challenges. To be practical, these applications need to be embedded in an educational platform that can deliver complete learning activities online and then assess student progress as they work through these activities. Teachers and educators should be able to customize the activities and assessments, so that they can tailor the learning strategies to the needs and interests of their students. The tools, models, and platforms needed to exploit this approach are sophisticated and probably too complex to be supported commercially. A better option is to make them open source and trust that their utility will support a broad community of users that will provide ongoing support and development.

OPEN SOURCE SOFTWARE

Open source software has revolutionized the creation, maintenance, and dissemination of large software packages. Open source software is defined by its unique copyright that states that anyone can use the source code without charge, providing that a user who improves the software must release the improved package with the same copy-

right. This means that the underlying code will always be available and free. There is not complete agreement on the best open source copyright, but the open source community has created a nonprofit to vet various licenses. OpenSource.org provides the best available definition of open source by approving a list of over a dozen open source licenses[1]. Source code released under one of these licenses can be called "Open Source."

A Short History of Linux

To understand the importance of the open source license and how it might improve education, it is important to understand its history and development.[2]

In the 1980's the Unix operating system was almost universally used in academic computing. At first, Unix had been copyrighted by AT&T but distributed free to academic users. Then AT&T started charging universities for Unix and many programmers felt betrayed because they had to pay for access to the Unix code, some of which they had written and freely contributed to the growing community of Unix users. In response, Richard Stallman organized GNU (pronounced GA-new), a project to make open source operating system components that extended the Unix operating system kernel.

Linus Torvalds freed open source from Unix by writing a Linux kernel and making it open source. Linux duplicates the Unix functions and interfaces to the GNU components. The resulting GNU/Linux open source combination has become known popularly as simply the Linux operating system. From its origins in reaction against corporate policies, it has grown to become the most popular operating system in the world. In an ironic twist, GNU/Linux is so important that it is now embraced by large corporations. For example, IBM delivers GNU/Linux in their computers and has invested billions of dollars in Linux software and services![3] Hewlett-Packard, Intel, IBM, NEC, and Oracle are only a few of the major companies now using Linux. These companies pay their own programmers to develop open source code and contribute to various collaborative open source projects that coordinate volunteer code development efforts.

The phenomenal popularity of GNU/Linux is due not only to the fact that it is free, but that it **better** that most alternatives: it supports a wider variety of hardware, is smaller, more reliable,[4] more easily extensible, more functional, and faster than other versions of Unix as well as the Windows operating system. The Macintosh OSX operating system is essentially a version of GNU/Linux. GNU/Linux is better because it is open source, allowing more talent to be recruited to develop code that even Microsoft can hire. As a result, a worldwide community of user-developers has sprung up that shares tasks, refines the software, and critiques ongoing work. The code that survives tends to be lean, elegant, and stable. Because it is open source, the code stays public and keeps getting better.

---

[1] http://www.opensource.org/licenses/

[2] The best history of open source is "Open Sources: Voices from the Open Source Revolution" available free online at http://www.oreilly.com/catalog/opensources/book/toc.html

[3] IBM Annual Report 2000, page 30.

[4] See http://www.gnu.org/software/reliability.html

This brief history of GNU/Linux provides lessons for how open source software can be produced. Perhaps the dynamic that has produced GNU/Linux can produce better educational applications.

Open Source Applications

The leaders of the open source movement, such as Stallman (2002) and Torvalds (Torvalds & Diamond, 2002), insist that open source is the best way to develop **any** kind of reliable and elegant software, and predict that most software will sooner or later be distributed as open source. Already the open source idea has reached far beyond operating systems. In addition to several browsers, there are now open source spreadsheets, word processors, a presentation package (like PowerPoint), and gimp, an image manipulation program that is compares to PhotoShop.

It is important to realize that open source applications do not need to be run under the Linux operating system. Schools could adopt a policy of using open source applications without needing to switch to Linux. Most new educational software is being written in Java because it is the best multiple-platform development environment. Software written in Java can run under Windows, Mac, and Linux operating systems, and others. Schools, with their legacy computers and operating systems, will want to use software applications that run under any operating system, and in most situations, applications written in Java will serve this need.[5]

The conventional wisdom has been that free software is unsupported, and therefore, a potential problem in the long run. Any software requires two kinds of resources: authoring and support. The authoring part is obvious, but too often the critical role of support is overlooked. Software without support is almost useless, because it cannot be fixed when a bug occurs due to a change in software the application needs. For instance, an application might be written in Java 1.4, which has a few incompatibilities with Java 1.5 and now all computers are upgrading to version 1.5. Or the operating system might be changed and some old calls are replaced by far better ones. Or some flaw in the application may not have been apparent until computers ran faster. The list of sources of problems is endless and it takes only one problem to render a package useless.

Unsupported software will generally loose functionality and eventually fail to work as software and hardware is upgraded. This is fatal for public domain software, but not for open source that is supported by a user community. It is important to realize that open source is not "public domain" software, which is software that bears no copyright. Public domain software may simply be the executable application, not its source code. If the source is not accessible, then there is no possibility of a community to form to improve it. But even if the source available for a public domain application, someone who makes some minor improvements can always copyright their improvements. This means that users of public domain software might at any time have to start paying for the software they thought was free. This threat of possible future loss of access to public domain software inhibits contributed improvements. As a result, public domain software remains frozen in time and dies away for lack of support. In contrast, open source is copyrighted, but in a way that ensures that any improvements made will be made available to all, so open source software can be supported and evolve.

---

[5] It happens that Java is itself open source, but that does not mean that programs written in Java must be open source.

Open Source and Business

At first blush, it would appear that free open source software is incompatible with commercialization and that companies would shun it. But on closer examination, it is clear that open source creates many opportunities for entrepreneurs. For instance, the simplest way to install GNU/Linux in your computer is to buy a GNU/Linux release from a company like Fedora.[6] You could find exactly the same material free on the Web, but the time saved by using their automatic installer that provides well-tested, compatible, documented, interoperable components is well worth the low price. GNU/Linux creates service and hardware opportunities that justify the huge investments that companies like IBM are making in open source. It is, therefore, possible that open source educational applications would support a variety of commercial educational services, that might, for instance, collect and vet applications, provide one-stop installations, connect applications to existing student management systems, provide consulting services, and offer teacher professional development that used the open source software.

Open Source Software in Education

The history of open source software suggests that a commitment to open source could help supply just the kinds of sophisticated software that commercial suppliers are generally unable to provide.

Why Does Open Source Work?

The idea of open source software seems to be utopian and unrealistic. The prevailing way of producing software is expensive, tightly controlled, and requires a hierarchical organization, epitomized by Microsoft. How could a group of volunteer, unpaid hackers with no schedule and almost no organization possibly produce anything of value? It defies logic. And, more importantly, if good software was produced, how would it be supported? Yet the huge growth of open source software is proof that these questions have been answered.

By examining successful open source software it is possible to discern how authoring and support have been provided. This provides clues about what characteristics are needed to produce and support educational open source software:

Broad use. Apache[7] filled a need for a Web server and is now the world's most popular one. At some point, each successful open source package reaches a tipping point at which there are enough people dependent on it so that the distributed users are willing and able to provide development and support services for it. The key is creating a base of users who are so committed to the package that they are willing to invest time and resources to keep it alive. This is more likely if the software is useful enough to solve the needs of many potential users.

Functionality. Most successful open source packages are large and provide important functionality. Anyone contemplating using open source faces a decision to

---

[6] http://www.redhat.com/en_us/USA/fedora/

[7] http://www.apache.org/

either use the free open source code or build an equivalent. Building something yourself is often very attractive: it's more fun, you control it, and your company can copyright it and make money from licenses. On the other hand, it is a risk to be dependent on a community of unknown hackers who make code that you do not fully understand. The savings that are realized by using open source can trump these disadvantages. If the open source code is powerful enough to save substantial time of expensive programmers, then the decision to use it instead of creating new software is easier.

Good design. A new user of open source software needs to be able to support and, in some situations, modify the software. This means that programmers need to understand the code well. Well-designed software is far easier to understand and support, because it is documented or self-documenting, logical, and compact. The authors of GNU/Linux have taken great pride in the quality of their designs. Good design also requires modularity; GNU/Linux, for instance, is a large collection of modules. This has made it possible for individuals and small teams who work independently to make important contributions.

IMPLICATIONS FOR EDUCATION

Successful open source software for education will need to share these characteristics; it should consist of well-designed, highly functional packages that can serve diverse needs. Small applications that are easily replicated such as a simple function grapher will not take off. Highly specialized software, even if well designed and highly functional, will fail as open source because of the limited audience willing to provide support.

There is already one example of an open source educational application that fits this profile. It fills a huge need, provides substantial functionality, and is well designed. OpenACS—an online course platform—started at MIT, morphed into a product called Ars Digita that attracted considerable funding in the late 1990's, went bust, and re-emerged as open source called dotLRN. OpenACS is a sub-set of dotLRN that is used by growing group of universities worldwide and supported by a community of companies.[8]

OpenACS solves a problem faced by almost every university worldwide and many other institutions, namely what technology to use for online courses. We started offering online courses a decade ago, first using Web pages, then waiting for a university-based package that never was completed, then using LearningSpace, which was later discontinued by IBM, and finally shifting to Blackboard. Every one of these platforms, and many others that we have investigated, has limitations that restrict the educational value of the materials delivered to students. Only when we shifted to OpenACS were we able to access the source code and make the upgrades that we needed. In so doing, we were doing just what is needed to help create a viable open source community: contributing new functionality that anyone can use, not because we felt generous, but because making these improvements were in our own self-interest.

The lesson from this example is that open source educational applications can thrive if the software solves an identifiable need faced by many users. It also illustrates the diffi-

[8] See http://openacs.org/community/companies/

culty commercial suppliers have in providing long-term support of large educational packages.

Diverse Expertise Needed

There is one other aspect of open source software that could be an impediment to creating and sustaining an educational open source community: the separation of those who write the code and those who use it. For open source systems development, these two communities are one. When open source was synonymous with system components, the programmers and users were the same—systems programmers made open source system components that they and their colleagues needed and used. As a result, the people writing the code were intimately aware of what was needed. When open source spread to applications like productivity and graphics tools, the programmers may not have known their audience, but they had successful commercial versions of the software as guides. In education, the separation between programmer and user is more complete and as a result, the software developers can become disconnected from the needs of education.

If you search the Internet for open source educational software, there are a surprising number of hits. But on closer inspection, most of what turns up is of two kinds: operating system components and primitive educational applications such as electronic flash cards or grade books. Neither can make a significant impact on education. Better operating system components might reduce the costs of educational computers. While that would be valuable, it still leaves unanswered the challenge of what to do with those computers. Primitive software that simply reinforces the educational paradigm of teacher-centered instruction based on student memorization of facts and algorithms will only impede needed change in education.[9]

Educational open source applications will find wide use only when experts from two distinct communities contribute: programmers and educational designers with content expertise. Since very few individuals can be at the cutting edge of both areas, it is unlikely that innovators working alone will create important educational open source software the way Linus Torvals created the Linux kernel. It is probably the difficulty of bringing these two areas of expertise together that accounts for the paucity of educational open source applications.

An Architecture for Open Source

We have been fortunate to be able to develop open source software in teams that contain some of the best educational programmers, science, technology, engineering and mathematics (STEM) content experts, and educational designers. Through various collaborations and formal centers, or work has been informed by the best thinking about software design, STEM content, and instructional design. The result of this work has been an outpouring of innovative technology-based instructional materials. The details of these materials are documented elsewhere (Tinker, 2004); what is important here is that while developing these materials, we have relied on an architecture that that allows programmers and content designers to work independently.

---

[9] For a review of open source educational applications, see (Tinker, 2005)

Guided Exploration of Models and Tools

Our architecture evolved through our analysis of the most fruitful contributions that technology can make to education. We see a huge need for tools and models that students can use for learning through guided exploration of important content in all educational levels and subjects.[10] Models and tools are what have fueled the general growth of computers, not only in academia, but also in business and government. A sophisticated component represents a huge investment in programming, but it is worthwhile because it can have broad application across disciplines and educational levels. We are certain that there are many more comparable models and tools waiting to be developed.

Our emphasis on guided exploration is important to our architecture: students learn best through exploration but are easily lost unless they are given guidance. Thus, the architecture must be able to control student ability to explore and must be able to provide context-sensitive guidance. The amount of guidance provided depends on your educational philosophy and can be tested by empirical research; the architecture can provide any degree of control or open-endedness desired.

Components, Platforms, and Learning Activities

We have found it valuable to use an architecture that separates software into *components*, *platforms,* and *learning activities*. The *components* are the models and tools that provide the content functionality that can be applied in many educational contexts. The *platforms* supports components by allowing an author to set the learning contexts, link concepts together, provide hints and guidance, and supply additional information. A platform can use one or more components, determine their initial conditions, and limit the available options. A platform can also accept student inputs and access student progress. The *learning activities* are specific student activities that have been authored using the platforms. For example, we developed BioLogica, a genetics *component* that incorporates the laws of Mendelian genetics and allows the student to explore reproduction at the molecular, cellular, individual, and population levels (Buckley et al., 2004). This component models classical genetics including linked traits and lethal genes, but it also supports mutations and environmental impact on populations over hundreds of generations, enabling the user to model genetic drift and evolution of new species. We made an earlier version of this component a stand-alone application that students could explore, but soon discovered that it was too sophisticated unless we also provided guidance to students that framed questions, suggested explorations, and eliminated options not needed for a particular investigation.

This led to the development of Pedagogica, a *platform* initially designed only to support the development, delivery, and assessment of student activities that used BioLogica (Horwitz & Christie, 1999; Horwitz & Tinker, 2001). Initially designed for the BioLogica component, it has been used with many other components. Pedagogica uses a scripting language–JavaScript–to control the use of any component in a learning activity. Each script creates a specific learning activity. Over fifty different *learning activities* have been

[10] The terms "tools" and "models" are subject to many interpretations. "Tools" is used here as software applications that are relatively content-free and give the user a useful set of functions, such as word processors, browsers, graphics editors, and geographic information systems. "Models" is used for interactive software that duplicates some of the characteristics of real or imagined systems, such as a flight simulator, SimCity, or the global climate.

developed for Pedagogica and various combinations of one or more component. Some of these activities are simply one page containing some text and a BioLogica model with some controls. Other activities are rich, branching, multiple-page explorations containing multiple models, different components, other multimedia options, and embedded student assessment. Each student activity consists of a JavaScript program. Because JavaScript is a complete programming language, almost any functionality can be built into a Pedagogica student activity. On the other hand, activity authoring is limited to JavaScript users.

A second example of the component-platform-activity architecture is provided by the Molecular Workbench (Berenfeld, Pallant, Tinker, Tinker, & Xie, 2004; Pallant & Tinker, 2004). The Molecular Workbench *component* is a sophisticated molecular dynamics model developed for education that models the interactions of large numbers of atoms, molecules, fields, and light. Students interacting with this component can gain deep insights about the atomic-scale world that can help explain a very wide range of phenomena typically studied in all the sciences, engineering, and applied technology. The MW component can be controlled by Pedagogica, but the development team wanted to expand the range of authors beyond script writers, so they developed an alternative *platform* called MW Pages which does not require programming. MW Pages is essentially a specialized word processor that accepts components such as MW and treats them the way a large text character would be treated. Input devices such as buttons and sliders and outputs such as graphs are also accepted by the word processor and logically linked to the MW component. Additional multimedia components can also be placed on the page, and pages can be linked to create large lessons. The resulting *student activity* is essentially a text document designed for this special word processor[11]. In this way, a student activity can be authored much the way any text can be generated, without the need to acquire any programming or scripting skills. As a result, a very wide range of teachers and staff members have contributed over 200 student activities.[12]

Lessons Learned

Several important lessons have been learned from the experience of developing these packages.

The component developer supplies the content–genetics and molecular dynamics in the examples. Developing a component requires content expertise, but not educational expertise, because the activity author using the platform can control the student experience. The content developer also can ignore all the functions provided by the platform and concentrate on building a great tool or model. Content experts typically include too many options and too much detail, especially for beginning students. In this architecture, the component developer can be unfettered because the activity author will decide what options and functions will be available to the user in any particular context.

The platform developer needs neither content nor educational expertise. The platform is the closest to a system component. It needs to interface with components

[11] See http://molo.concord.org for a database of learning activities based on MW.

[12] BioLogica, Pedagogica, the Molecular Workbench, MW Pages, and numerous smaller packages used in our work are all open source, licensed under the LGPL copyright.

and support authoring. A full-function platform might also handle inter-component communication, student registration and passwords, activity loading, caching, persistence, student assessment, media, and many other tasks required for delivery and use of student activities in an environment where connectivity is intermittent.

The components and platforms fit the criteria for successful open source packages. They both have a high level of functionality and, potentially, broad use. The hundreds of learning activities based on BioLogica and MW components and platforms attest to their generality and the tens of thousands of current student users give a sense of their potential utility.

Student learning activities could be commercialized, because they can be copyrighted. All the educational design is contained in the student learning activities and these are analogous to documents produced by a word processor: the documents can be copyrighted, even if the word processor is open source. So far, because we have been funded from government grants, we have chosen to make our activities freely available in the spirit of open source, but there is no requirement that this be true in the future. A publisher, for instance, could use components and a platform to develop proprietary learning activities. Very high quality activities, keyed to a text, might be expensive to develop and need to be protected so that the publisher can recoup the development and support costs. This would be desirable because it would expand the community of self-interested users dependent on the open source component and tools, increasing the likelihood that these will be supported and improved.

CONCLUSIONS

The separation of components, platforms, and learning activities could be critical to the creation of valuable, open-source applications that begin to fulfill the educational promise of technology. This separation reflects the different skills needed to produce effective educational materials: systems experts to build platforms that can handle the generic issues raised by the educational context, content experts to develop the components, and educational experts to develop the actual activities that students see. By separating the development effort into these compartments, we ensure that each kind of expert can work independently of the others. This division of responsibility has been important in the history of GNU/Linux and could be equally important in educational applications.

Only a few components and platforms are needed, but they are large packages that require substantial programming resources. The investments they represent could be made by an open source collective because the components and platforms have very broad potential applications. Educational developers in both the for-profit and non-profit worlds could use the platforms and the components they support to create huge libraries of specific learning activities. If authoring is as simple as using a word processor, non-technical faculty and teachers could customize existing activities to fit the needs of their specific curriculum and students, generating even more use of these resources.

The components and platforms developed at The Concord Consortium serve as important examples of the value of this architecture. They have already provided the envi-

ronment for the development of large numbers of learning activities that are in wide use. We believe that they are ready for the kind of broad adoption that would result in a viable open source community. We would be delighted if others began using, supporting, and improving them. There are many other tool and model components that could be developed to increase the value of these packages.

We continue to develop the platforms and components and to develop ever-increasing numbers of learning activities based on these. We are currently adding the ease of authoring developed by MW Pages to Pedagogica and we are adding new functions to MW. We are working with colleagues at the University of California, Berkeley to develop SAIL, a framework for platforms that will include tools that simplify the development of new platforms.

Whether the current generation of components and platforms developed at The Concord Consortium achieve widespread use is less important than the architecture that they represent. Sophisticated tools and models and the ability for students to learn about them through guided exploration is essential to realize the changes that current technology makes possible. There does not seem to be the funding from either the private sector or government agencies to fund the development of these large, sophisticated applications. The only alternative is to turn to the open source development model, and the only way this can work is to divide the programming tasks so that individuals and small teams can make meaningful contributions. It appears that the division of the effort into platforms, components, and learning activities is a reasonable and logical way to achieve this.

Genetics, atomic interactions, and many other phenomena are the same in every country. Students everywhere learn best through guided exploration. Therefore, there is a global need for the kinds of components described in this paper. If nations worldwide could contribute to a common effort to develop and support these, we will realize the educational promise of technology far sooner with far better software that is free and open source.

BIBLIOGRAPHY

Berenfeld, B., Pallant, A., Tinker, B., Tinker, R., & Xie, Q. C. (2004). *From genetic code to protein shape using dynamic modeling*. Paper presented at the Proceedings of the NARST 2004 annual meeting, April 1-3, Vancouver, BC, Canada.

Buckley, B. B., Gobert, J. D., Kindfield, A. C. H., Horwitz, P., Tinker, R., Gerlits, B., et al. (2004). Model-Based Teaching and Learning with BioLogica: What Do They Learn? How Do They Learn? How Do We Know? *Journal of Science Education and Technology, 13*(1), 23 - 41.

Feurzeig, W., & Roberts, N. (Eds.). (1999). *Modeling and Simulation in Science and Mathematics Education*. New York: Springer.

Gobert, J., & Buckley, B. (2000). Introduction to model-based teaching and learning in science education. *International Journal of Science Education, 22*(9), 891-894.

Horwitz, P., & Christie, M. (1999). Hypermodels: Embedding curriculum and assessment in computer-based manipulatives. *Journal of Education, 181*(2), 1-23.

Horwitz, P., & Tinker, R. (2001). Pedagogica to the rescue: A short history of hyper-models. *@CONCORD, 5*(1), 1, 12-13.

Pallant, A., & Tinker, R. (2004). Reasoning with atomic-scale molecular dynamic models. *Journal of Science Education and Technology, 13*(1), 51-66.

Stallman, R. M. (2002). *Free software, free society: Selected Essays of Richard M. Stallman*. Boston: The Free Software Foundation.

Tinker, R. (1990a). Computer Based Tools: Rhyme and Reason. In E. F. Redish & J. S. Risley (Eds.), *Computers in Physics Education*. Reading, MA: Addison-Wesley.

Tinker, R. (1990b). Modelling and theory building: Technology in support of student theorizing. In D. L. Ferguson (Ed.), *Advanced Educational Technologies of Mathematics and Science* (Vol. 107, pp. 91-114). Berlin: Springer-Verlag.

Tinker, R. (2004). Free computer-based learning resources. *@CONCORD, 8*(2), 1, 4-5.

Tinker, R. (2005). Freeing Educational Applications. *@CONCORD, 9*(1), 10-11.

Torvalds, L., & Diamond, D. (2002). *Just for fun: The story of an accidental revolutionary*. New York: HarperCollins Publishers.