6-2008

Improving the Computer Science in Bioinformatics Through Open Source Pedagogy

John David N. Dionisio
Loyola Marymount University, dondi@lmu.edu

Kam D. Dahlquist
Loyola Marymount University, kdahlquist@lmu.edu

Improving the Computer Science in Bioinformatics Through Open Source Pedagogy

John David N. Dionisio
Department of Electrical Engineering & Computer Science
Loyola Marymount University
Los Angeles, CA 90045, USA
dondi@lmu.edu

Kam D. Dahlquist
Department of Biology
Loyola Marymount University
Los Angeles, CA 90045, USA
kdahlquist@lmu.edu

Abstract: Bioinformatics relies more than ever on information technologies. This pressures scientists to keep up with software development best practices. However, traditional computer science curricula do not necessarily expose students to collaborative and long-lived software development. Using open source principles, practices, and tools forms an effective pedagogy for software development best practices. This paper reports on a bioinformatics teaching framework implemented through courses introducing computer science students to the field. The courses led to an initial product release consisting of software and an *Escherichia coli* K12 GenMAPP Gene Database, within a total "incubation time" of six months. (1)

Categories and Subject Descriptors: K.3.2 [Computers and Education]: Computer and Information Science Education — *computer science education*
General Terms: Design, Management
Keywords: Bioinformatics education, Curriculum development, Open source, Pedagogy, Scientific computing

1. SCIENTIFIC COMPUTING AND THE DIGITAL DIVIDE

Bioinformatics is just one example of how disciplines beyond computer science and engineering are, more than ever, relying on computational techniques and information technologies to conduct research and apply its results. This trend can place undue pressure on practitioners in those fields, who frequently have to rediscover, relearn, or keep up with work in the computer science and software development realms in order to get the most out of their work. These practitioners may have little to no formal training in these domains, resulting in unnecessary (and sometimes unknowing) repetitions of past discoveries and errors that could have been avoided through the use of current best practices in software development [21]. In some cases, this "update burden" is abandoned, and tools or paradigms that are viewed as out-of-date in computer science and software engineering remain in use in other disciplines. These practitioners then fail to maximize the potential of the computational power and technology that is available to them. At worst, software flaws slow or impede research. For instance, numerous researchers were affected by the retraction of five papers published over a six-year period due to the discovery of software flaws [13].

This "digital divide" impacts research using scientific computing, but it has its roots in education — after all, tomorrow's scientists are today's students. If these students are to be prepared adequately for careers in research and industry, then today's curricula must find ways to help close this technological gap. One of the fields for which this need is most pressing is biology, where mathematical and computational skills are required to answer the questions of 21st century biology [20]. However, addressing the digital divide requires more than simply combining traditional biology and computer science coursework to create a bioinformatics degree (described by Altman [2]): pedagogical practices in computer science itself may be disconnected from the expectations and skill sets required of computer scientists in industry or interdisciplinary research groups (Table 1). Computer science undergraduates typically work alone instead of in a team, produce isolated programs from scratch instead of large modular projects, and throw away their code after the assignment has been graded instead of maintaining and improving it over an extended period of time [8].

Baxter et al. responded to the issue of the digital divide with a list of suggested best practices for software development in bioinformatics research [4]. These practices include up-front project design, program and process documentation, quality control, data standards, and project management. Note, from Table 1, how these best practices are missed by the traditional computer science curriculum, yet correspond to real-world expectations in computer science industry or research. Baxter et al. give examples of project management software and large bioinformatics projects that incorporate these principles. With one exception, all of these tools and applications are *open source*.

Table 1. Disparity between traditional computer science curriculum [8] and software development best practices in industry/research [4].

Traditional computer science curriculum	*"Real world" expectations (industry or research)*	*Best practices for software development*
Students work alone	Members work together as teams	Project management
Isolated programs	Large modular projects	Up-front project design
Throw away code after grading	Code maintained over an extended period of time	Program & process documentation; quality control; data standards

The open source connection is no coincidence — open source principles (a) foster the best practices recommended by Baxter et al., and (b) fulfill the real-world expectations of computer science industry and research practitioners [3]. Open source and bioinformatics make for a particularly natural fit, since this research community has been at the forefront of the open source movement (e.g., Open Bioinformatics Foundation [16]). Would open source, then, hold keys to closing the digital divide in fields such as bioinformatics?

This paper reports on an effort to explore precisely that question. Initially implemented in two graduate courses, "open source pedagogy" has resulted in a new way to teach bioinformatics, an open source project, XMLPipeDB, that evolved from initial conception to a released product in six months [22], and a master's thesis that built upon the project to produce new versions and releases [15].

2. OPEN SOURCE: FROM SOFTWARE TO PEDAGOGY

The open source movement has begun to influence computer science education [10, 11]. David A. Patterson, past president of the Association for Computing Machinery (ACM), suggests leveraging open source software and best practices for coursework [18]. *Recourse*, a curriculum development project at Loyola Marymount University (LMU) that is partially funded by the National Science Foundation, is currently integrating open source elements into the computer science curriculum [8]. The work described in this paper was performed within the context of that project.

The open source culture is defined by a set of software criteria (the official Open Source Definition [17]), community values derived from these criteria, and software development practices and tools (Table 2 [5, 9, 19]). From the perspective of the framework proposed by Baxter et al. (Table 1 [4]), up-front project design, program & process documentation, and project management are addressed by the cultures' sense of accountability and community, which results in continuous integration and test-driven workflows [5, 9]. Quality control emerges from the sense of responsibility that accompanies the rights provided by open source licenses. Finally, the availability, modifiability, and longevity of source code facilitate data standards.

The middle column of Table 2 indicates how these values and best practices have implications for bioinformatics pedagogy. Source code becomes the concrete artifact representing an effort to solve an authentic problem with realistic complexity — a cornerstone of science curricular reform movements such as problem-based [1, 6] or case-based learning [14]. Nothing is thrown away, remaining available for future students and the open source community in general. Sufficiently large problems require team effort [12]; thus, source code, by faculty and students alike, must reside in a centralized, public repository. Everyone's work becomes visible for code review or debugging. Quality is emphasized: students are compelled to document and perform automated tests on their code. An added benefit of this open source pedagogy is that it facilitates long-term course projects beyond the current semester and class.

Table 2. Concordance between open source values [8], an active learning pedagogy [1, 6, 14], and open source software development practices and tools [5, 9, 19].

Open Source Values	*Active Learning/ Bioinformatics Pedagogy*	*Open Source Practices & Tools*
Source code is available, modifiable, and long-lived	Authentic problem to solve with realistic complexity	Central code repository; version control; provenance of code
Accountability to a developer and user community	Participatory and collaborative work; peer review	Task and bug trackers; continuous integration; test-driven workflows
Responsibilities accompany rights	Responsibility and ownership of the learning process	Documentation: in-line, user manual, Wiki

3. IMPLEMENTATION IN BIOINFORMATICS COURSES

Computer Science 698/Biology 498: Special Studies in Bioinformatics was an experimental course team-taught by a biologist (K.D.D.) and a computer scientist (J.D.N.D.). Enrollment in Spring 2006 consisted of eight students from LMU's graduate program in computer science. Several students concurrently worked in industry; none of them had more than college-level introductory biology, and most had not taken biology since high school. Instead of attempting to survey the broad field of bioinformatics topics [2], we decided to focus on biological databases and to develop software to address a need in this area. This project became XMLPipeDB, a reusable, open source tool chain for building relational databases from XML sources [22, 23].

The topics covered in the course were driven by the needs of the software project. The biologist began with a brief introduction to the entire field of bioinformatics to put the later work in context, and the students were charged with studying examples of several different biological databases. The computer scientist discussed the relational data model, XML data sources, XML-object-relational mapping, and modeling languages. Finally, the students were introduced to an existing bioinformatics software package, GenMAPP (Gene Map Annotator and Pathway Profiler), a tool for viewing and analyzing genomic and proteomic data on biological pathways [7]. The goal of the XMLPipeDB project was to facilitate the creation of new GenMAPP gene databases for species that were not currently supported by the software. This problem is authentic, serving a research need, and was of sufficient complexity due to the ongoing bioinformatics issue of reliably relating gene identifiers.

Throughout this project, students were expected to uphold the best practices advocated by Baxter et al. [4]. The students were asked to perform up-front project design and program and process documentation. Quality control came in the form of in-class code reviews and bug tracking. The project itself utilized XML data standards and was managed by the instructors with cycles of design reviews, setting of milestones, and evaluation of results. Each student chose their own development environment (e.g., Eclipse, NetBeans, text editor + *ant*, etc.) but worked as a team from a SourceForge-hosted repository [23].

Because of the initial success of the bioinformatics course, a summer session course entitled *Open Source Software Development Workshop* followed, which covered the following topics: the open source definition, open source licenses, open source development practices and tools, version control, test-driven development, continuous integration, bug/task tracking, development life cycles, documentation, community participation and roles, comparison between open- and closed-source projects, and the economic impact of open source development. The workshop format used classroom meetings as coordination and update sessions, with much of the actual work taking place outside the classroom.

Four students continued XMLPipeDB development in this course. Three of these students were in the prior bioinformatics course, while one of the students was new to the project, thus modeling the entry of new members into an open source community. The "new member" reviewed XMLPipeDB's existing documentation and source code while prior members continued with software development. Eventually, the new member became a committer as well, finishing the course by contributing some refinements and a Web site "first draft" for the project.

4. SUCCESSES, CHALLENGES, AND RECOMMENDATIONS FOR BROADER IMPLEMENTATION

XMLPipeDB development led to its first release, a gene database for the *Escherichia coli* K12 bacterium that is formatted for use with the GenMAPP application [7], approximately six months after the inception of the first Special Studies in Bioinformatics course. Development continues along two tracks: (1) applications and libraries relating to gene databases and (2) end-user data sets culled from various sources and transformed or formatted using these applications and libraries. The project has also led to a masters' thesis that sought to automate the process of updating and maintaining GenMAPP gene databases based on external XML sources that are themselves updated and modified by third parties over time; over the course of this work, the code was also refactored and debugged as a whole [15]. Software development continues and is visible on SourceForge [23], with news, updates, and documentation posted on the project's own Web site [22].

The challenges faced in teaching this course mirrored the real world challenges of interdisciplinary research groups. As opposed to more traditional courses in computer science, we found as instructors that we spent as much time managing the students as we did on the actual technical content of the course. Although some students took on a leadership role, we found that most of the students — even those with industry experience — had brought their expectations of what the course would be like from undergraduate experiences in a traditional curriculum and preferred to interact with the faculty rather than each other for questions and direction, despite our exhortation that they treat their classmates developing modules downstream from them as customers or clients. To foster increased communication, accountability, and a sense of team amongst the students in the follow-up workshop course, we demanded more rigorous adherence to the project management tools (task and bug tracking) and implemented weekly reporting on a Wiki. This shifted some of the responsibility for project management back to the students, again preparing them for real world expectations.

Another challenge lies in project continuity, particularly in the area of support and maintenance: XMLPipeDB, as well as future open source projects that are initiated under the *Recourse* methodology, are essentially student-run, and thus subject to heavy turnover at the undergraduate and masters levels. Many software projects — open source or otherwise — face turnover as well, and generally this is addressed by documenting the project from multiple perspectives (introductory, tutorial, reference, etc.) at a sufficient level of detail. With this in mind, the instructors designated both a users' manual and a group paper on the project's functionality and its history of design and implementation decisions as course deliverables; for the "new member" who joined during the summer session course, building a project Web site effectively introduced this student to the work and tested the quality of the documentation that was already available.

This documentation feeds directly into assessment and evaluation of student work. When team projects become direct components of student coursework, then what is the most appropriate means for evaluating individual student performance within these projects? In this teaching framework, a number of instruments were established for an objective, systematic evaluation of student work. The computer science instructor performed code reviews. Correlating functionality to primary contributor(s) was facilitated by repository annotations of who committed which section of code. The students were also graded on the aforementioned users manual and group paper by both the biologist and computer scientist. Review by the biologist ensured that the documentation could be understood by a non-specialist and that the students had delivered on the project goals, again a hallmark of interdisciplinary research groups. Furthermore, the students themselves were asked to evaluate the group paper and the project's source code as a whole — a form of peer review. Finally, the students submitted individual "statements of work" that detailed their contributions to the project and a self-evaluation of the quality of their work. Thus, as with other aspects of the project, the students were held responsible for their own assessment and evaluation. We found that peer evaluations of student work closely matched instructor evaluations. Overall, both the project content and the process were a collaborative and open effort. And despite the fact that students had varying levels of knowledge and skills coming into the course, even the weakest student contributed usable code. In addition, students reported that they were attracted to the research area of bioinformatics and that they learned practices that they would take back to their industry jobs.

5. CONCLUSION

A framework for teaching bioinformatics based on active-learning pedagogy and on open source principles, values, and practices has been successfully implemented at LMU. Students (a) participated in a real research project that produced actual, working software and datasets, (b) participated in collaborative, team-based learning as goes on in real industry and academic research, and (c) took responsibility and felt a sense of ownership for the project and their own learning. While the project focused on the area of biological databases, this method can be extended to address the needs for teaching the theoretical foundations and algorithms of bioinformatics. The particular courses mentioned in this paper included only graduate students, but the method is also being implemented with undergraduates as part of a broader curriculum improvement project.

ACKNOWLEDGMENTS

We would like to thank Barbara Marino and M. Catharine McElwain for the opportunity to team-teach these courses. We would also like to thank John Jungck for his insight and suggestions on reporting our work.

REFERENCES

[1] Allen D and Tanner K (2003). Approaches to cell biology teaching: learning content in context — problem-based learning. *Cell Biology Education* 2:73-81.

[2] Altman RB (1998). A curriculum for bioinformatics: the time is ripe. *Bioinformatics* 14:549-550.

[3] Barnett L and Schwaber CE (2004). Applying open source processes in corporate development organizations. Technical report, Forrester Research, Inc., Cambridge, MA. http://vasoftware.com/sourceforge/request_info-dl.php?paper=9.

[4] Baxter SM, Day SW, Fetrow JS, and Reisinger SJ (2006). Scientific software development is not an oxymoron. *PLoS Computational Biology* 2:e87.

[5] Beck K, Gamma E, and Saff D (2006). JUnit test infected: programmers love writing tests. http://junit.sourceforge.net/doc/testinfected/testing.htm.

[6] BioQUEST curriculum consortium. http://www.bioquest.org.

[7] Dahlquist KD, Salomonis N, Vranizan K, Lawlor SC, and Conklin BR (2002). GenMAPP, a new tool for viewing and analyzing microarray data on biological pathways. *Nature Genetics* 31:19-20.

[8] Dionisio JD, Dickson CL, August SE, Dorin PM, and Toal R (2007). An open source software culture in the undergraduate computer science curriculum. *ACM SIGCSE Bulletin* 39(2):70-74..

[9] Fowler M (2006). Continuous integration. http://martinfowler.com/articles/continuousIntegration.html.

[10] Howland JE (2000). Software freedom, open software and the undergraduate computer science curriculum. http://www.cs.trinity.edu/~jhowland/ccsc2000/ccsc2000/ccsc2000.html.

[11] Kegel D (2003). The undergrad CS program, Linux, and open source. http://www.kegel.com/linux/edu/curriculum.html.

[12] Linder SP, Abbott D, and Fromberger MJ (2006). An instructional scaffolding approach to teaching software design. *Journal of Computing Science in Colleges* 21:238–250.
[13] Miller G (2006). A scientist's nightmare: software problem leads to five retractions. *Science* 314:1856–1857.
[14] National Center for Case Study Teaching in Science. http://ublib.buffalo.edu/libraries/projects/cases/case.html.
[15] Nicholas J (2006). GenMAPP Builder 2.0: A System for the Creation of GenMAPP Gene Database Files. Master's thesis, Department of Electrical Engineering & Computer Science, Loyola Marymount University, Los Angeles.
[16] Open Bioinformatics Foundation (2006). http://www.open-bio.org.
[17] Open Source Initiative (2006). The open source definition. http://www.opensource.org/docs/osd.
[18] Patterson DA (2006). Computer science education in the 21st century. *Communications of the ACM* 49:27–30.
[19] Raymond ES (2001). The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary. O'Reilly Media.
[20] Steen LA (2005). Math and Bio 2010: Linking undergraduate disciplines. *The Mathematical Association of America*.
[21] Wilson GV (2006). Where's the real bottleneck in scientific computing? *American Scientist* 94:5–6.
[22] XMLPipeDB home page. http://xmlpipedb.cs.lmu.edu.
[23] XMLPipeDB SourceForge project site. http://sourceforge.net/projects/xmlpipedb

ENDNOTE

(1) Partial support for this work was provided by the National Science Foundation's Course, Curriculum, and Laboratory Improvement Program, Award No. 0511732.