Teaching Software Engineering Including Integration with Other Disciplines

Richard M. Stillman
rstillma@nova.edu
School of Computer and Information Sciences
Nova Southeastern University
Fort Lauderdale, FL 33314, USA

Alan R. Peslak
arp14@psu.edu
Information Sciences and Technology
Penn State University
Dunmore, PA 18512, USA

Abstract

Software engineering is Money Magazine's top rated profession. The development of novel information systems has created new industries and catapulted developers to wealth and stardom. Yet, for many students of computer and information systems, software engineering is just another hurdle they must jump to satisfy degree requirements.

How best to teach software engineering so that students appreciate its unique and vital lessons remains an unanswered question. Our software engineering course exploits students' experience in specific domains as a foundation for learning the skills of software development. The course syllabus provides a vehicle for honing one's development skills, practicing abstraction, and finally experiencing the "aha" phenomenon when the student has successfully integrated two different fields of knowledge into a new discipline. We report the results of this approach.

Keywords: Higher education, software engineering, information systems, active learning environment, domain knowledge

1. INTRODUCTION

No one would have predicted: that an efficient search algorithm would form the foundation of an immensely profitable company; that software to enable peer-to-peer transfer of copy written music would become available on-line (or that this technology would subsequently be deemed illegal); that the entire genetic code of several species including homo sapiens would be sequenced and available on-line, leading to a new generation of biology researchers working without a brick-and-mortar laboratory.

Software engineering is Money Magazine's top rated profession (Kalwarski, 2006). The development of novel information systems has created new industries and catapulted developers to wealth and stardom. Yet, for many students of computer and information systems, software engineering is just another hurdle they must jump to satisfy degree requirements.

How best to teach software engineering so that students appreciate its unique and vital lessons remains an unanswered question.

Hazzan (2007) points out that mastering software engineering requires the ability to deal with "soft ideas", concepts that elude formal definition. Soft ideas come into existence when the programmer is thinking about a domain apart from the software. Dealing with soft ideas is a skill that cannot be explicitly taught; they have to be experienced to be understood. To create a novel system, a programmer must almost instinctively feel the connection between an untapped domain and the power he knows a computer system can bring to that domain.

Software engineering courses often fail to convey to students the importance of the topic they are teaching. Students tend to believe that success in building information systems requires just the technical know-how to write code. Henry and LaFrance (2006) stress the importance of active learning by engaging students in relevant projects. Petcovic et al (2006) note that the globalization of software requires graduates to have experienced reasonable simulations of the complexities of real-world software development. Grisham et al (2006) go so far toward real-world simulation as to intentionally leave the project requirements vague, so that the student must take responsibility for this fundamental step in the development process.

Promising new realms of endeavor often spring from the unlikely combining of separate disciplines. Evolutionary algorithms and bioinformatics are two compelling examples. Ali (2006) suggests multidisciplinary software engineering projects, as for example a software engineering student partnering with an architecture student to create 3-dimensional building visualization software.

Myers (2007) observes that in order for a software engineering project to be really educational, it must be a substantial endeavor not a "toy" application, despite the limited time available; and it must involve a meaningful domain.

The flash of insight that leads a visionary to introduce computer technology to a new domain requires creativity, knowledge of the domain and of the technology, and, perhaps most of all, the ability to think abstractly. Kramer and Hazzan (2006), summarizing a workshop on The Role of Abstraction in Software Engineering, note that the participants agreed that abstraction should be taught in software engineering courses, but cautioned that abstraction "seems to be a talent-laden skill: some will get it, many will not, and a few will be very good at it."

Whether the application is advertising, multimedia, or molecular biology, the development of novel and useful software requires that the developer integrate software engineering with specialized knowledge of another, unrelated discipline.

Several other researchers have noted the importance of domain knowledge to successful software engineering. Falbo, Guizzardi, and Duarte (2002) suggest that domain knowledge is essential to software reuse. Maidantchik, Montoni, and Santos (2002) observe that complex software systems require iterative development as the team masters understanding of the domain. Robillard (1999) in the *Communications of the ACM* suggests "Software development is the processing of knowledge in a very focused way. We can say it is the progressive crystallization of knowledge into a language that can be read and executed by a computer. The knowledge crystallization process is directional, moving from the knowledge application domain to software architectural and algorithmic design knowledge, and ending in programming language statements."

Thus, in real-world software engineering, the application of domain knowledge is the starting point for software engineering projects.

2. METHODS

In our master's level software engineering course, we assign the prototypical exercises, but encourage the student to respond to these exercises using a domain for which the student possesses specialized knowledge or interest.

Specifically, the student is asked to perform each of the following steps of software development for a sizable existing or imagined system in one or more domains of his or her choice:

1. Outline the process that you would use to build the system.
2. Write a statement of scope.

3. Create a functional decomposition, estimate LOC, effort, and cost. (In the case of an existing system, this question can be answered using what the student knows about the domain to reverse engineer the software.)
4. Give examples of pertinent data abstractions and the associated procedural abstractions.
5. Use code from your chosen domain to illustrate examples of cohesion and coupling.

At this point, the student is asked to do the following major projects:

1. Produce a comprehensive proposal for a major software project involving a domain of your choice. The proposal should include information such as Project Overview & Scope; Process & Project Management; Requirements Analysis & Design; Feasibility Analysis; Coding, Testing & Maintenance; Project Plan & Scheduling; Risk Management; Ethical & Legal Considerations; Delivery & Documentation; Conclusions
2. Develop a working prototype of a portion of the system that you proposed.

3. RESULTS

Many of our master's students are professionals. They represent a variety of industries. Therefore, it is not surprising that we received submissions covering a reasonably wide range of domains. Here are some examples:

Example 1

A manager for a major railroad company developed a proposal to rail shippers, third-party logistics companies, and shipping brokers for a rail visibility and supply chain management application.

Example 2

A software contractor to the US Army developed a three tier application used to update the airfield approach maps that are used on the U.S. Army utility and cargo helicopter flight simulators instructor operator stations.

Example 3

A lead senior client server analyst for a major cruise line with twenty years of experience in the industry developed a three tier Cruise Line Client Reservation System.

Example 4

A student with a strong background in biological science proposed the development of a web-based information system for a bioinformatics laboratory. We use this student's work as an example of the type of project submitted. Appendix 1 provides the full table of contents of the project.

The informational flow model in Figure 1 and the scope and boundary diagram of Figure 2 illustrate scope of this ambitious project.

The 5-year cost of system development is estimated at $750,000. The prototype submitted is a browser-based HTML/JSP client layer with a Java Servlet architecture connecting to the database layer via a number of problem domain and data access classes. The database layer is a hybrid of a Microsoft Access relational structure for internal data and seamless data access class connectivity to public databases of biological data. The data-flow diagram in Figure 3 and the entity-relationship diagram in Figure 4 show some of the functionality that will be required for this prototype.

Figure 5 is a screen print that shows a portion of the actual functionality of the prototype system submitted for this project.

Student Feedback

The goals of our master's level software engineering course are to teach the development of software-intensive systems, software quality factors, software engineering principles, system life-cycle models and paradigms, requirements definition and analysis, behavioral specification, software design, implementation, software testing techniques, verification and validation, system evolution, and software project management.

End of semester evaluation forms were reviewed to determine the perceived efficacy of the software development assignment described in this paper.

Feedback from students enrolled in the three semesters in which this project was assigned

revealed uniform agreement that this approach met the course goals. An example of the type of feedback was that the approach was a "comprehensive, well-organized, academically-enriching experience".

4. DISCUSSION

There is little controversy that really perfecting the skills required for successful software engineering requires an active, hands-on process. The question is how to optimize the didactic experience. Our approach of having the student select a familiar domain for the final project has both benefits and limitations.

Benefits

One can assign a specific task for a student or a group of students to complete during the semester, and perhaps that approach would better simulate an industrial environment. But permitting self-selection of the domain confers the following benefits:

Respecting the Student's Talents: The student can focus on the process of software development, and is relieved from the need to study an unfamiliar domain. The student's energies are channeled into the practice of system development. Van der Duim et al (2007) include *respecting students' diverse talents* as a best practice in software engineering education.

Future Benefits to the Student and the Industry: The project itself may provide a springboard for the student's entry into system development involving his or her own profession. This may confer immediate benefit to the student as an employee, the employer, and even the industry. Bernhart et al (2006) feel that teaching software engineering requires that the project framework should reflect real-world applications. Turhan and Bener (2007) go even further: they recommend, "simulating a chaotic environment" so that students' expectations will match the reality of software development.

Potential Benefit to the Instructor: As an interesting side effect, the professor expands his general knowledge of areas that information systems development may benefit.

Overcoming Some Obstacles

There are downsides to this approach. But we believe these obstacles can be overcome.

Coordination of Group Projects: For obvious reasons it is difficult to coordinate a group project that allows students to select the project domain. On the other hand, when students collaborate on a project for which only one member of the group has the domain expertise, the resulting process becomes a reasonable simulation of real-world software engineering.

The Instructor's (Possibly Limited) Knowledge of the Chosen Domain: In the absence of in-depth knowledge of a particular domain, the professor may have a bit of difficulty helping the student should problems occur during system development, and then evaluating the resulting deliverables. The former reflects a real-world concern inherent in system development, but in practice this is overcome routinely nonetheless. The latter has not been a problem in our experience because the student's plan, protocol, and prototype form a unit that, when analyzed together, can be verified by checking for internal consistency, and of course the code can always be checked for appropriate functionality. Furthermore, there is a safety valve: students are required to obtain approval for their project plan at the beginning of the course. This gives the instructor an opportunity to request a change of plan if really necessary. Finally, we emphasize that we utilize this approach only in graduate-level courses.

Summary

In summary, we teach software engineering by facilitating integration of the system development process with a domain of particular interest to each student. The syllabus provides guidelines, but each student creates his own assignment. We believe that this approach simulates the process by which technological ingenuity drives the emergence of new fields.

As philosopher-scientist Edward O. Wilson (1998) observed, "… asking the right question is more important than producing the right answer. The right answer to a trivial question is also trivial, but the right question, even when insoluble in exact form, is a guide to major discovery."

5. ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their insightful suggestions.

6. REFERENCES

Ali, M. R. 2006. Imparting effective software engineering education. *SIGSOFT Softw. Eng. Notes 31*, 4 (Jul. 2006), 1-3.

Bernhart, M., Grechenig, T., Hetzl, J., and Zuser, W. 2006. Dimensions of software engineering course design. In *Proceeding of the 28th international Conference on Software Engineering* (Shanghai, China, May 20 - 28, 2006). ICSE '06. ACM Press, New York, NY, 667-672.

Falbo, R. d., Guizzardi, G., and Duarte, K. C. 2002. An ontological approach to domain engineering. *In Proceedings of the 14th international Conference on Software Engineering and Knowledge Engineering* (Ischia, Italy, July 15 - 19, 2002). SEKE '02, vol. 27. ACM Press, New York, NY, 351-358.

Grisham, P. S., Krasner, H., and Perry, D. E. 2006. Data Engineering education with real-world projects. *SIGCSE Bull. 38*, 2 (Jun. 2006), 64-68.

Hazzan, O. 2007. The influence of software intangibility on computer science and software engineering education. *SIGSOFT Softw. Eng. Notes* 32, 3 (May. 2007), 7-8.

Henry, T. R. and LaFrance, J. 2006. Integrating role-play into software engineering courses. *J. Comput. Small Coll.* 22, 2 (Dec. 2006), 32-38.

Kalwarski, T., Mosher, D., Paskin, J., Rosato, D. (2006, May) . Fifty best jobs in America. *Money 35*(5): 94-101.

Kramer, J. and Hazzan, O. 2006. The Role of Abstraction in Software Engineering. *SIGSOFT Softw. Eng. Notes* 31, 6 (Nov. 2006), 38-39.

Maidantchik, C., Montoni, M., and Santos, G. 2002. Learning organizational knowledge: an evolutionary proposal for requirements engineering. In *Proceedings of the 14th international Conference on Software Engineering and Knowledge Engineering* (Ischia, Italy, July 15 - 19, 2002). SEKE '02, vol. 27. ACM Press, New York, NY, 151-157.

Myers, J. P. 2007. A web emphasis in software engineering. J. *Comput. Small Coll. 22*, 4 (Apr. 2007), 268-274.

Petkovic, D., Thompson, G., and Todtenhoefer, R. 2006. Teaching practical software engineering and global software engineering: evaluation and comparison. *In Proceedings of the 11th Annual SIGCSE Conference on innovation and Technology in Computer Science Education* (Bologna, Italy, June 26 - 28, 2006). ITICSE '06. ACM Press, New York, NY, 294-298.

Pressman, R. (2006) *Software Engineering: A Practioner's Approach* Sixth Edition McGraw-Hill, New York.

Robillard, P. N. 1999. The role of knowledge in software development. *Commun. ACM 42,* 1 (Jan. 1999), 87-92.

Turhan, B. and Bener, A. 2007. A Template for Real World Team Projects for Highly Populated Software Engineering Classes. In *Proceedings of the 29th international Conference on Software Engineering* (May 20 - 26, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 748-753.

van der Duim, L., Andersson, J., and Sinnema, M. 2007. Good Practices for Educational Software Engineering Projects. In *Proceedings of the 29th international Conference on Software Engineering* (May 20 - 26, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 698-707.

Wilson, E.O. 1998. *Consilience: The Unity of Knowledge*. Vintage Books, Inc., New York.

FIGURES

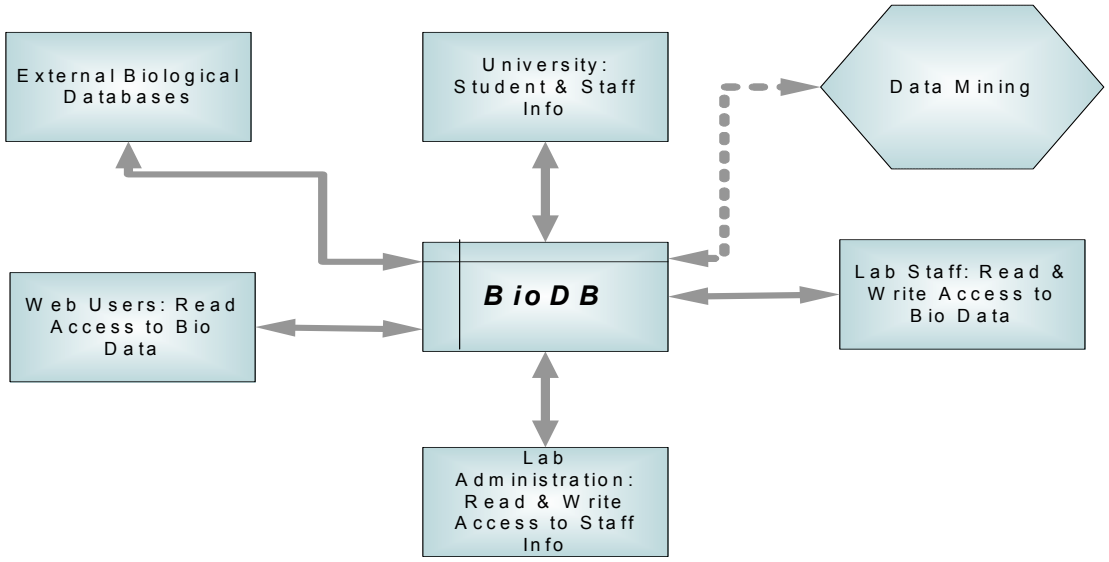

Figure 1. The informational flow model of a student's proposed bioinformatics information system

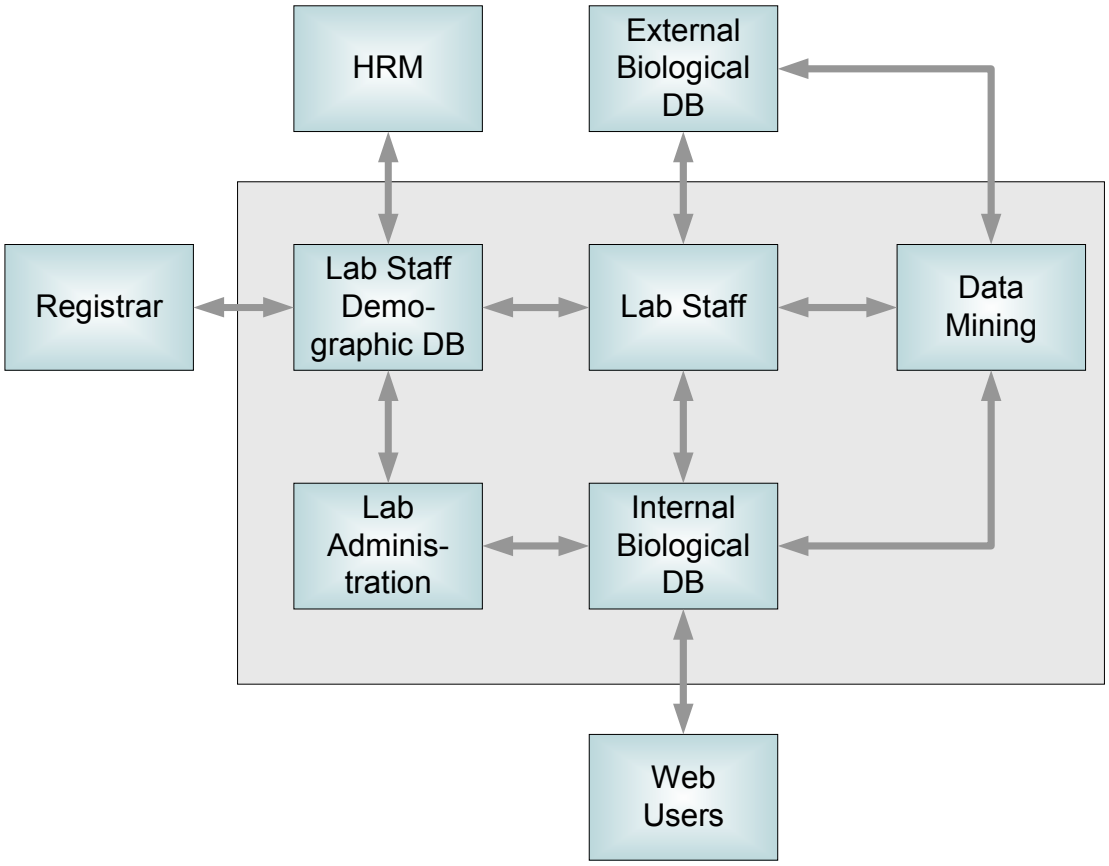

Figure 2. A scope and boundaries diagram demonstrating a novel approach to combining a university's information system with a laboratory's data generation and AI capability

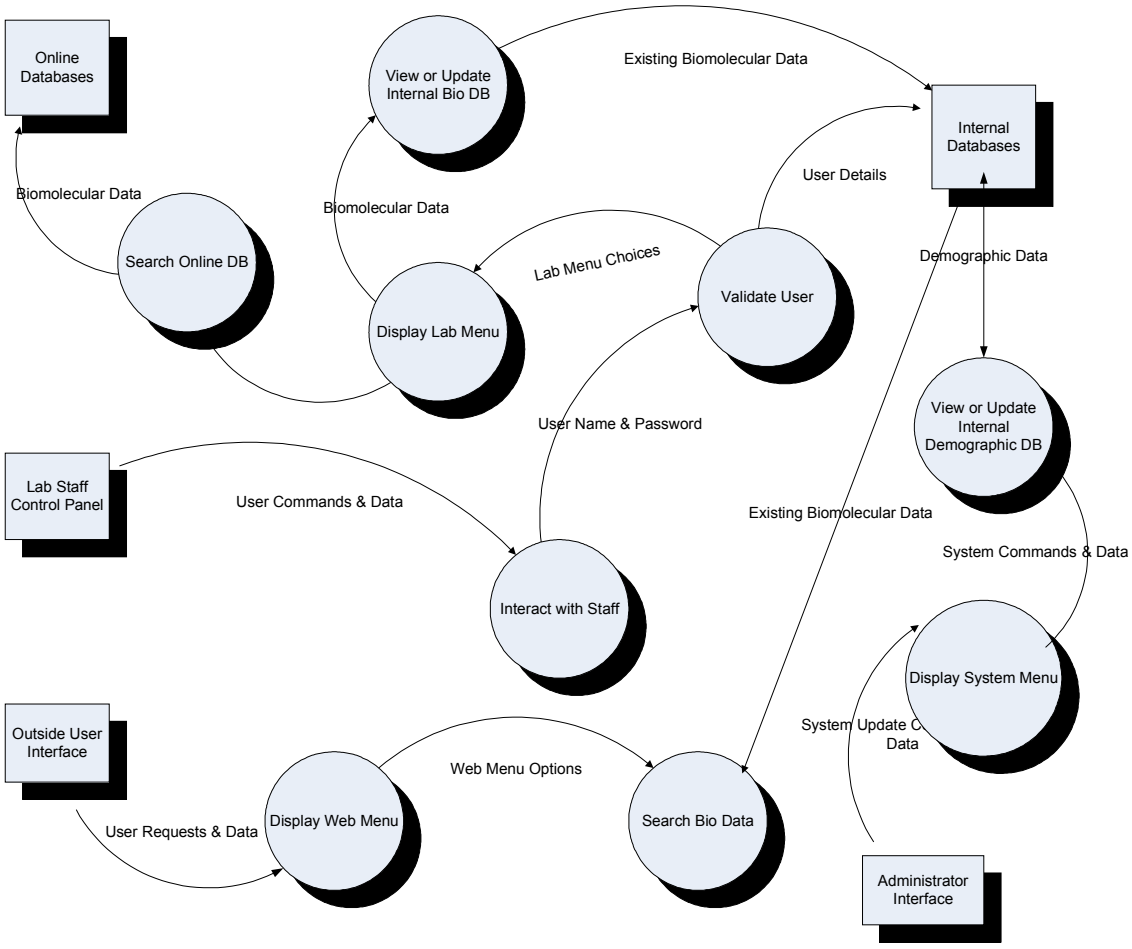

Figure 3. A data-flow diagram of a portion of the system. If he didn't appreciate it before, the complexity of engineering a system of this magnitude is now apparent to the student. Software engineering is more than just writing code.

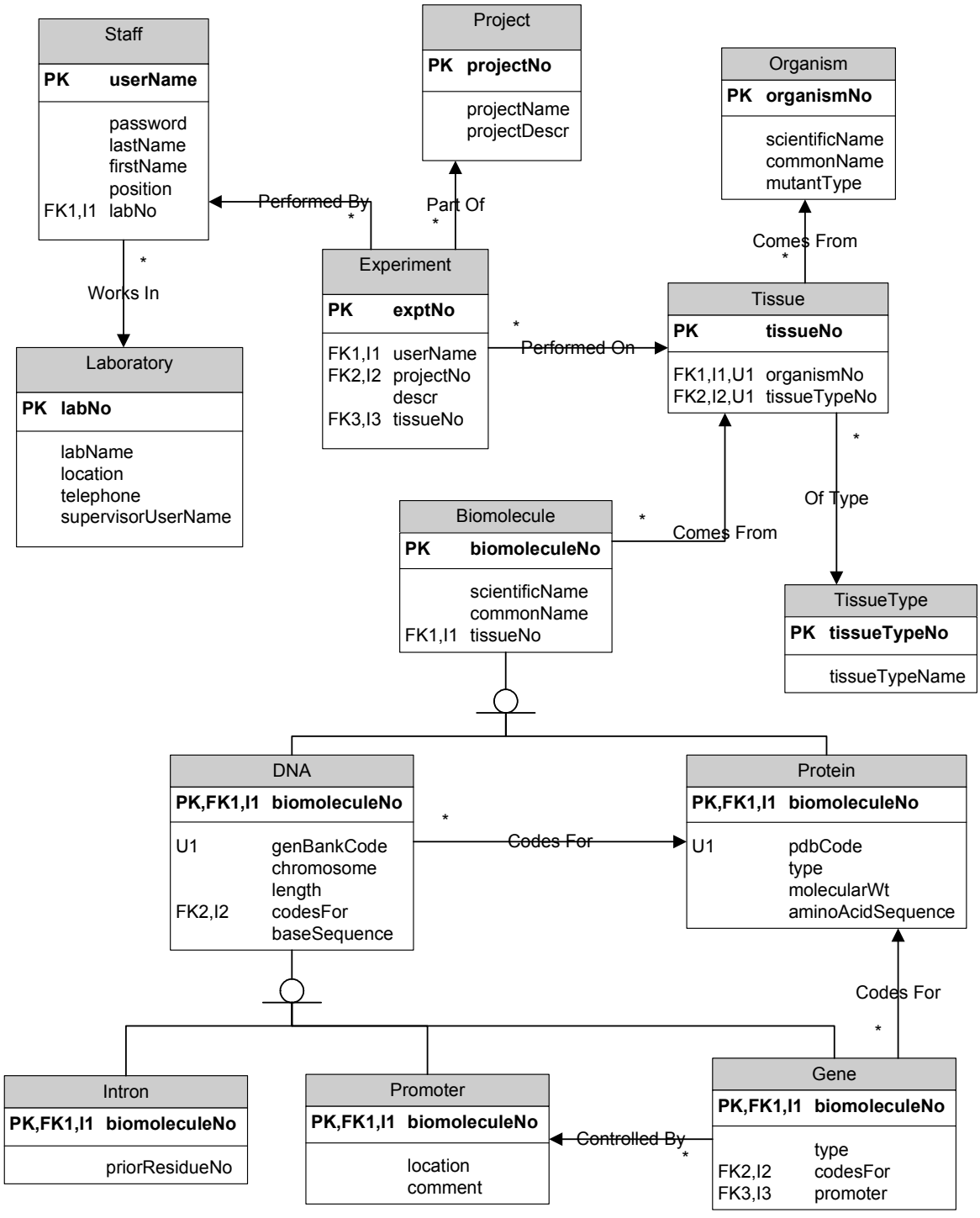

Figure 4. An entity-relationship diagram of the portion of the system the student has implemented in the prototype.

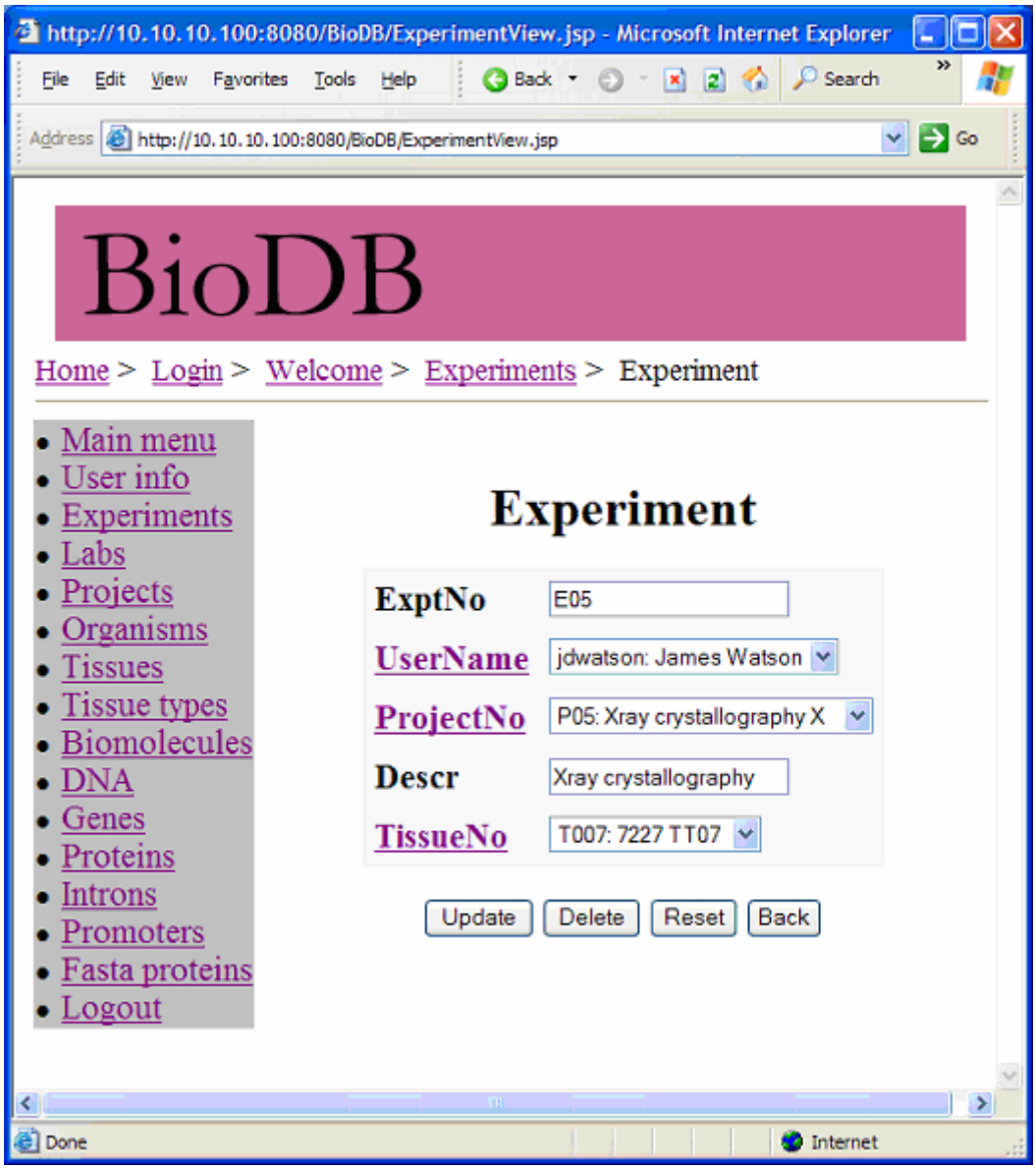

Figure 5. A screen print of the web interface to the student's prototype system. The code behind this page utilizes a variety of technologies including HTML, Javascript, and JSP.