IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4720

Integrating Artificial Intelligence and Software Engineering: An Effective Interactive AI Resource... ...does more than teach AI

Stephanie E. August
Electrical Engineering and Computer Science Department
Loyola Marymount University
One LMU Drive MS 8145
Los Angeles, CA 90045-2659, USA
saugust@lmu.edu

ABSTRACT

When teaching artificial intelligence (AI) courses, instructors may feel that our undergraduate programs are successful when the students have demonstrated understanding of their coursework—often by developing much of the project code from scratch. This submission contends that this is not sufficient and that we can (and should) instill sound software engineering development practices as well. Such an approach can enable students to build upon the prior work of others and create software upon which others can build. An approach to develop an effective interactive AI resource, which would facilitate learning in AI as well as engender sound software engineering practices is discussed.

INTRODUCTION

An effective undergraduate program in computer science must balance theory and practice, while reinforcing within each course the lessons learned in other courses. An effective resource that is developed to teach search techniques, for example, would not only walk students through both uninformed and informed search techniques, but would also promote software engineering standards and development methodology by presenting the design of a software module before presenting code and by adhering to established coding standards with regard to both coding style and documentation of code within all code modules. It would support courses in algorithms by including components that guide students in analyzing the complexity analysis of the algorithms being presented. In addition, the resource should make available reusable code modules that can be incorporated into larger applications, thereby making students aware of the importance of defining and managing component interfaces and supporting projects that can build upon previous efforts, rather than always starting from scratch.

One of the interesting features of our undergraduate program at Loyola Marymount University is a two semester senior sequence in software engineering. During the first semester of senior year, the computer science students design and implement a large group project, following standard software engineering methodology. During the second semester, students complete a smaller individual project following the same methodology. Often, to create an interesting project or make their applications behave "intelligently", the students need to fill in gaps in their knowledge of artificial intelligence, and, in almost all cases, they code everything from scratch. As a result, we see the same types of projects with the same levels of implementation each semester. We decided to improve this situation by creating an interactive, reusable AI resource facility. Toward this end, we are in the process of developing an on-line Library of Artificial Intelligence (AI) Resources that facilitates the reuse of AI algorithms and applications in undergraduate projects. The objective of LAIR is to provide a set of modules that can be used to teach concepts in an undergraduate introductory AI course. Good software engineering practices will be instilled in students by carefully designing and documenting the modules, which will subsequently be used as building blocks to improve the performance of and provide additional functionality in software projects assigned in both AI and non-AI courses.

This paper briefly discusses some of the challenges inherent in teaching AI and computer science in general and outlines our approach to meeting these challenges. We conclude with a review of related work, an outline of our evaluation methodology, and a report on our accomplishments to date.

THE CHALLENGES

One of the challenges of designing programming assignments for an AI course is to balance the benefits of understanding the intricacies of complex algorithms against the benefits of being exposed to a variety of algorithms and applications (Wyatt, 2000) (Kumar, 2001). LAIR proposes to address this issue by presenting concepts as a set of interactive exercises. Because less course time will need to be devoted to the implementation of algorithms, more time will be available to cover artificial intelligence concepts and applications. Students will have more opportunity to explore a wider range of topics than would be possible if a large portion of the course were devoted to the complex programming techniques needed to implement these algorithms. We expect that this broad base of experience will put students in a better position to take on large AI projects in graduate courses, where the students do have a need to explore topics in depth.

A challenge to every computer science course is to instill good software engineering (SE) practices in students (Pour, 2000) (Lethbridge, 2000). LAIR will reinforce concepts taught in SE courses by constructing exercises and their solutions using the format for software development that is employed in our software engineering courses, and by adhering to established coding standards in all software included in the project. Coded solutions to the exercises will be available to students to enhance the functionality and performance of class projects developed for other courses, promoting the concept of open-system architectures. In our program, our experience has shown that such a resource would be useful for projects in AI, SE, and database management systems courses.

A challenge to the field of computer science in general is to attract and retain women. Women often do not fit the stereotype of code-writing- and implementation-detail-oriented software designer ("hacker") to whom courses are often geared, and tend to regard a computer more as a tool for solving problems than as a "toy". (Frenkel, 1990) suggests that women are more likely to remain in the field when courses are geared less toward code writing and more toward code reading and developing mental models of the processing involved. In addition, structured labs with exercises that can be completed before leaving class build the students' sense of accomplishment and confidence. LAIR's emphasis on structured design, exemplars, and structured exercises addresses this problem.

Our university is a teaching institution, and places a high value on integrating material from one course with material presented in another, both within and across disciplines. LAIR supports this tradition by encouraging collaboration among computer science courses. The lessons, while geared toward AI, will view concepts from many perspectives. For example, analysis of the complexity of the algorithms included in the resource will reinforce lessons related to computer science theory, and exercises related to data storage will be used to teach data structure concepts in an undergraduate database management (DBMS) course.

LAIR will make it easier for the students in both the AI and the software engineering courses to build onto existing applications. The resulting projects can subsequently be developed into additional repository components and made available for future development. Over a period of time, this will allow us to add more advanced topics to the electives available to students. In addition, it will provide a basis for graduate student study and more comprehensive projects than would be possible otherwise.

APPROACH

We have identified nine components that are essential to achieving our goal of creating an effective interactive AI resource. Our choice of components is patterned after standard software development methodology and the components used in the programming exercises that accompany (Winston, 1992). The components include:

The Idea. This is a brief explanation of the concept to be presented, similar in content to a project proposal and requirements analysis. It will explain what the program segment is expected to do, without describing how to implement it. This component will also contain references to additional sources of information on the topic.

Sample Input/Process/Output. An annotated trace of the program in execution, showing system input and describing the processing taking place will give the student a clear idea of how the algorithm behaves. This component corresponds to a concept of operations and user manual for the implemented concept.

Implementation-independent Design Description. This design description will give the student an idea of how the concept can be implemented from a functional perspective. It will include both textual descriptions and interactive diagrams, such as a unified modeling language (UML) description that allow the user to explore the design in a top-down manner, revealing as much design as the developer chooses to expose and enabling the student to see specific examples of design patterns that might be relevant to their problem. This description will remain programming language-independent not only for clarity, but for platform portability and longevity as well. The programming languages, platforms, and operating system versions we use to implement our algorithms change much more rapidly than the ideas we are implementing! The design description will identify areas in the design that reflect significant trade-offs or design variations so that students can better understand a component's behavior characteristics. In our resource environment, we are exploring ways to facilitate the means to notate this. Our approach is similar to capturing variation points that are part of product-line architecture development (Kerner, 2000) (Northrop, 1999), but is novel in its application to AI problem domains. One approach we plan to investigate is exploring design variation descriptions by defining architectural aspects that can be searched later. We hope to exploit special tags in UML to define these aspects.

Implementation-specific "HINT" files. Each of these files will contain part of the code needed to implement the concept in a particular programming language, as well as HINTs that guide the developer in implementing the remainder of the code. These HINT files are similar to those used with a great deal of success in courses offered by information technology training companies. We will also provide procedures for compiling and running the student's implementation of the code. The file will include file header comment blocks and follow an established coding style. This will reinforce the notion that coding standards do exist and should be adhered to whenever code is generated. For aspect-oriented components, we are currently looking into ways that HINTs can be defined and managed as aspects[1]. We plan to conduct experiments on how effectively we can exploit aspects in retargetting different AI designs for different student projects. An additional feature, which is beyond the scope of this project but which would be particularly useful to implement, would be a preprocessor or code evaluator that compares the code that the student generates in response to the HINTs to the code that is needed to implement the missing pieces.

Test Suite and Driver. Students need to be reminded that a program is not complete until it has been successfully tested. We will provide an implementation language-dependent test driver with trial input and expected output to verify that the code has been correctly implemented. The student can then compare the expected results with the results generated by their code. Procedures for running the test will be included for each driver.

Experiments. The experiments will suggest ideas for enhancements and extensions to the basic code provided, and go beyond the testing outlined for the test driver above. Interactive demonstrations of the concept will be available to allow the student to experiment with various forms of input. The HINT files, implementation-specific test drivers and implementation-specific experiment drivers will be repeated for each source code implementation provided (described below). For example, there might be an aspect-oriented Java version, a C++ version, and a Common Lisp version of an implemented concept.

Applications. Descriptions of real world applications that use this concept answer the question "Why do we need to learn this? Where will we ever use this?" Related references and animated, graphical representations of algorithms, such as a soccer-playing robot demonstrating autonomous software agent architecture, will put the concept in perspective, and provide to the interested student additional avenues to explore on the topic.

Complexity analysis. This complements the work done in an algorithms class, and should point out the different ways that complexity can be measured for a particular problem and explore the respective tradeoffs.

Source Code. Solutions to the exercises in HINT files will be available for students to check their work, conduct experiments, and use as off-the-shelf building blocks. The test suites and drivers can be used with these solutions to perform *black box* testing of the code. Links to other implementations as well as more comprehensive implementations can also be provided.

User Interface Offering Multiple Perspectives

In order for the resource to be useful to a broad spectrum of users, we plan to provide multiple interfaces for the users. A student in an introductory programming course might simply need to know how to interface with a piece of code to a complete simple project, while a student in an introductory course in artificial intelligence might need a more detailed web-based tutorial view of the resource, and a knowledgeable student in a project class who wants to build upon existing code modules would need to access all available source code in archive format that is easily downloaded without first wandering through the related lessons. Our approach is to store the components (lessons, hint files, documentation, source code) in a database, and use web pages to draw the needed information from the database as needed. The overhead of the

database should pay off in ease of maintaining content and multiple accesses levels, and will enable us to maintain the presentation logic independent of the data itself. Since design artifacts such as UML, aspect-oriented architectural tags, HINT descriptors, and documentation notes can be managed via XML, we are looking into developing XML schemas that can integrate our nine component elements. XML provides a powerful way to exchange information between the various supporting applications we are designing for the resource environment and eventually other institutions.

RELATED WORK

Interactive websites can be invaluable in bridging the gap between textbooks and applications. Textbooks provide useful information, yet they are limited in their ability to convey concepts that are subtle, inherently dynamic, and often complex, which characterizes many ideas in AI. There is a growing interest in the AI community to develop effective, interactive, AI resources (Effective, 2001). A website that showcases these efforts can be found at http://mainline.brynmawr.edu/EIAIR/. LAIR is being developed in conjunction with this effort, yet is unique in its emphasis on software engineering and standards and the explicit inclusion of a design component with each concept included in the resource.

One of the limitations of existing AI code repositories such as the Carnegie Melon University site http://www.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/areas/0.html, the American Association of Artificial Intelligence website http://www.aaai.org/Pathfinder/pathfinder.html, and those associated with AI textbooks such as (Russell and Norvig, 1995) and (Winston, 1992), is that the code is rarely accompanied by design descriptions, rarely includes detailed structured exercises to help students implement the concepts, and does not follow a consistent documentation standard. LAIR will make a valuable contribution in that regard.

There is an increasing need within the computer science community to instill in students an appreciation for standard software engineering methodology, and to narrow the gap between vision, education, and standard practice in the software development community (Pour, 2000) (Lethbridge, 2000). This is not an issue in the defense industry, when clear guidelines for software developments are available and standards are strictly enforced. However, it is becoming an increasingly important issue in the "dot com" era when a significant amount of code is being developed without explicit requirements for standard software engineering practices to be followed. As a result, systems that are being developed that are poorly designed, undocumented, and difficult to maintain.

Evaluation Methodology

Our objectives are twofold: to provide resources to those who wish to augment their projects with AI algorithms, and to improve our students' grasp of fundamental AI algorithms and their use. We anticipate that the algorithms will be more widely used by students in our project-based classes and that the students in the AI classes will gain a deeper understanding of the algorithms and their relative merits through the availability of the resource. The effectiveness of LAIR in meeting these objectives will be measured using both quantitative and qualitative methods. Students in the undergraduate and graduate AI courses will be tested in class on their knowledge of the algorithms at the end of each term, with the results compiled and compared over the next four years. In addition, we will introduce LAIR to students in our group and individual software engineering projects at the beginning of each term, and conduct roundtable discussions of its usefulness at the end of each term in these classes. These discussions will explore the extent to which the students were able to incorporate the algorithms into their projects, the impact of the resource in terms of both design and implementation, and how we can make the resource more useful in the future. We will use the feedback from these discussions to modify the user interface to and content of LAIR to improve its utility over the life of the project. In addition, students in the software engineering courses will be required to include a written summary of their experience with LAIR in their final project reports and describe in detail which LAIR components contributed to the success of their projects. These reports will be summarized and used to track the utilization of the resource beyond the AI class itself. The ultimate test is whether the materials are useful to students in interdisciplinary courses, such as the multimedia project class and the new media workshop, and whether the effort results in well-designed projects that can in turn be built upon in subsequent years.

Current status

Last year we completed a proof-of-concept project that developed a framework for the repository of reusable AI code modules. This year we plan to implement the database and prototype web interface for LAIR, and continue to develop our content. The resource will store the exercise components in a relational database to facilitate the maintenance and security of the components. A web-based interface to the components will offer users easy, yet controlled access to the components, an environment for completing the exercises, and links to related resources on the Internet. The production version of the repository will be hosted on a set of four servers. Access to the repository will be available over the web. We estimate that it will require six months to complete the component for each topic. Given sufficient staffing, we expect to complete two to four topics each year over the next three years. Our first focus will be on search algorithms, then on neural networks, conceptual clustering, and genetic algorithms. Follow on work will include creating both natural language processing and knowledge representation tutorials that can be used in the AI course to introduce these topics and enhance other projects. Incorporating aspect-oriented approaches in new domains is still a promising area of research (Walker, 2002). Applying these approaches to support our AI resource environment is a new trend in which automated management of highly complex algorithms, designs, and other architectural information can be leveraged to improve not only the AI education but also the software development experience of our students.

REFERENCES

Communications of ACM. October 2001.

"Effective interactive artificial intelligence resources workshop program working notes." Russell Greiner, Chair. **Seventeenth International Joint Conference on Artificial Intelligence,** Seattle, 2001.

Frenkel, Karen A. (November 1990) "Women & Computing." **Communications of the Association for Computing Machinery, 33**(11), p. 34-46.

Kerner, Judy. (February 23-25, 2000) "Architectural Evaluation for Product Lines," **Proceedings of the Ground Systems Architecture Workshop 2000.**

Northrop, Linda. (March 3-5, 1999) "Developments in Product Lines and Architecture Evaluation," **Proceedings of the Ground Systems Architecture Workshop 1999**.

Kumar, Deepak. (January 2001) "How much programming? What kind?" Curriculum Descant, **ACM Intelligence, 12**(1), .

Lethbridge, Timothy C. (May 2000) "What Knowledge is Important to a Software Professional?" **Computer**, p. 44-50.

Pour, Gilda; Griss, Martin L.; and Lutz, Michael. (May 2000) "The Push to Make Software Engineering Respectable." **Computer**, p. 35-43.

Russell, Stuart J., and Norvig, Peter. **(**1995) **Artificial Intelligence: A Modern Approach**. Prentice-Hall, Englewood Cliffs, NJ.

Walker, Mark G., and August, Stephanie E. (2002) "Applications of Aspect-Oriented Software Development Techniques to Spacecraft Ground System Software." Submitted for publication.

Winston, Patrick Henry. **(**1992) **Artificial Intelligence.** 3rd edition. Addison-Wesley, Reading MA.

Wyatt, Richard. (2000) "Interdisciplinary AI." Curriculum Descant, **ACM Intelligence, 1**(2), Spring.