Increasing AI Project Effectiveness with Reusable Code Frameworks: A Case Study Using IUCBRF[*]

Steven Bogaerts and David Leake
Computer Science Department
Lindley Hall, Indiana University
Bloomington, IN 47405, U.S.A.
{*sbogaert, leake* }@*cs.indiana.edu*

Abstract

Instructors' ability to assign artificial intelligence programming projects is limited by the time the projects may require. This problem is often exacerbated by the need for students to develop significant system infrastructure, requiring them to spend time addressing issues which may be orthogonal to the AI course's core pedagogical goals. This paper argues that such problems can be alleviated by basing coding assignments on paradigm-specific *frameworks*, collections of reusable code designed to be extended and applied to a variety of specific problems. In addition, frameworks can provide a basis for further student research or application of projects to real-world domains, providing additional motivation. This paper illustrates the application of a framework-based approach to teaching case-based reasoning (CBR), introducing the Indiana University Case-Based Reasoning Framework (IUCBRF), discussing its design, and presenting sample exercises that take advantage of the framework's characteristics.

Introduction

The use of course projects has long been advocated in education, both as a way to enhance traditional courses and as the basis for new project-based instruction strategies (Blumenfeld & Soloway 1991). Studies support these approaches by suggesting that projects may have important benefits in forming more flexible understanding with improved transfer to real problems (for example, see (Boaler 1997)). Making the projects themselves deal with real-world domains may further enhance student motivation. However, attempts to apply such strategies can cause frustration for instructors and students alike, as they attempt to manage project complexity in the time available for course assignments (Marx *et al.* 1997; Krajcik *et al.* 1998). One reason for this is that real-world domains and full-scale systems may require significant detours into side topics outside the desired course focus (Barron *et al.* 1998; Blumenfeld & Soloway 1991). For artificial intelligence programming projects, significant detours include implementing algorithms and infrastructure needed for the function of a system, but conceptually orthogonal to the pedagogical goals of the course. Thus methods are needed for decreasing the effort which AI students must devote to these tasks.

In AI, a number of strategies have been used to maintain focus on the core issues. One strategy is to rely on high-level discussions of components in the abstract, enabling coverage of many central ideas. Unfortunately, such discussions provide poorly grounded abstract exercises rather than hands-on experiences and real experimentation. Another is to present demonstrations of particular algorithms, without requiring student coding, as in numerous web applets for AI. These can spur initial interest and give students an impression of the function of the algorithms, but make limited contributions to student's ability to implement AI systems.

Another alternative is for students to develop sample systems within simplified "toy" domains or to experiment with adapting code from simplified versions of real systems, such as Schank et al.'s microprograms (Leake 2002). Such tasks can be useful, but students whose experience comes solely from simplified systems may be unconvinced of the power of the methods and of the real-world value and scalability of their work. If small-scale approaches are used exclusively, students miss the boosts to self-efficacy and self-confidence that are often found in student accomplishments in real-world projects (Thomas 2000).

Thus, ideally, AI instructors should be able to provide students with opportunities to examine real-world domains in addition to toys, and to use fully-functional systems, with minimal overhead. This paper argues that the use of large-scale *frameworks* can help achieve this goal. A framework is a collection of reusable code designed to be extended and applied to a variety of specific problems (Johnson & Foote 1988). Frameworks can provide a basis for students to develop system components and perform experimentation in the context of a well-documented "real-world-strength" system that is easily understandable, adaptable, and extendible. Frameworks help focus students' work on building components that directly address key concepts, on a provided infrastructure that facilitates application to large-scale projects.

The paper begins by discussing the nature of frameworks and their benefits for educational use. It then examines a specific use of the framework-based approach to teaching AI: teaching CBR with the Indiana University Case-Based Reasoning Framework (IUCBRF), a newly-released, freely-

[*] We thank the anonymous reviewers for helpful comments.

available CBR framework designed for educational and research use (Bogaerts & Leake 2005). Using IUCBRF, a full CBR system for a real-world domain can be developed without necessarily requiring intricate student understanding or time-consuming construction of every component. This frees the instructor to focus on components of interest, at the desired level of coverage. The framework benefits instruction both through its *support for system development for CBR projects*—by providing standard component implementations, minimizing dependence between system components, and minimizing component dependence on the particular domain—and through its *built-in support for experiments* to study the effects of design decisions. This paper illustrates the use of IUCBRF in instruction with sample exercises exploring some key aspects of case-based reasoning.

Teaching Computer Science With Frameworks

Two properties of frameworks that make them useful for teaching are their provision of basic infrastructure and their minimization of dependencies, both between components and to particular domains. These properties greatly reduce the time required to build a complete system, and allow introductory students to use complete systems before they understand each component. Thus, instructional time may be used more effectively on, for example, covering additional topics and/or addressing real-world domains.

Providing Infrastructure

For many computer science topics, projects require much implementation of non-lesson-critical infrastructure before the issues of interest can be examined. Such work can provide good learning experiences for students, but can take time away from intended course topics. By providing this infrastructure to students (perhaps leaving that material for a more suitable course), frameworks allow students to quickly focus on the important issues and to perform experiments which might not be feasible starting from scratch.

Minimizing Component Dependencies

A natural way to study alternative approaches for any topic in computer science is to perform a comparative study of the behaviors of systems using the approaches. To perform such a study, the system must be designed to allow component genericity: components should not make assumptions about implementation details of other components, so that a different approach for a component can be swapped in without system-wide consequences.

Achieving genericity in student projects might require considerable attention to object-oriented design. While object-oriented design is worthy of in-depth study, it is tangential to the stated focus of comparing approaches for a particular AI task. Furthermore, student frustration at the time spent on incidental tasks may prompt them to rely on substandard designs and error-prone "hacks" to accomplish the desired genericity. The deficiencies of the result could severely limit student desire and ability to return to their systems for future coursework or research.

The framework design philosophy of independence of components addresses this issue. Neither the instructor nor the students must spend extra time to build such a design from scratch. They may instead proceed immediately to the stated focus of the work.

Minimizing Domain Dependencies

Ideally, student projects are not isolated efforts, but instead part of a growing toolkit of methods that students will apply to new domains in continued coursework or research. However, students who have built their past systems from scratch may face the arduous task of working with a system that has been hurriedly designed to satisfy the requirements of a particular domain, attempting to untangle these dependencies to implement a new domain. Again, unless great care was taken in the original design, such dependencies are likely to be numerous, complex, and undocumented, and so students may once again be tempted to attempt a quick error-prone fix to complete the assignment.

Frameworks designed to minimize domain dependencies can alleviate this problem. Such frameworks can represent the domain explicitly in a central location, enabling components needing domain-dependent details to simply ask the relevant object for these details, rather than requiring them to be coded in the component directly. Only on rare occasions should a component need such domain-specific knowledge that additional effort will be required. This assists students working with a framework to apply a single system to many domains.

Framework Construction

We expect the use of frameworks to be an effective teaching strategy across a wide range of AI areas. Although in some respects CBR may be seen as especially well-suited to a component-based approach, any AI method may be analyzed for its major tasks, categories of operations for the task of interest, and parameters for each step. Likewise, tasks can be analyzed for the major algorithms, components and data containers involved. These determine the access points of the framework, for providing customizability.

The properties described above provide broad guidelines to follow in designing a framework for any AI method: providing infrastructure, minimizing component dependencies, and minimizing domain dependencies. Providing infrastructure means not only providing the components specific to the topic of interest, but also more mundane components such as GUI widgets and experimentation support. Minimizing dependencies can be accomplished by careful separation and encapsulation of components into logical chunks that connect with other components only through general algorithm- and domain-independent interfaces. Finally, in very general terms, the implementation should be planned in a careful object-oriented manner (for example, see (Gamma *et al.* 1995)). The following case study provides a concrete illustration of how we have applied some of these design principles for case-based reasoning.

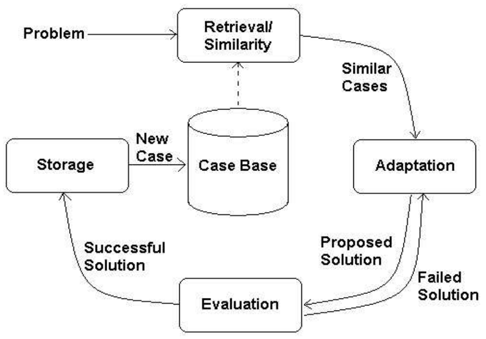

Figure 1: The CBR cycle

CBR and IUCBRF

CBR is an AI problem-solving approach in which a system stores prior problem-solving experiences (cases) and retrieves and adapts them to suggest a solution for a current problem. Students can be introduced to the area through textbooks (e.g., (Kolodner 1993) or on-line overviews (e.g., (Leake 1996; Aamodt & Plaza 1994)).

The basic CBR cycle, as sketched in figure 1, proceeds as follows. A problem is described using the vocabulary of the system. Similar past cases are retrieved, and are then adapted to propose a solution to the current problem. That solution is evaluated and either returned for more adaptation, or stored in the case base for use in future problems.

IUCBRF is a freely available open-source framework, written in Java, to facilitate the development of CBR systems. For a complete foundational discussion and technical details of IUCBRF, and for information on how to obtain the source code, please see (Bogaerts & Leake 2005). One of the goals of the IUCBRF project is to enhance CBR instruction with the benefits common to frameworks for a classroom setting, including provision of basic infrastructure, and minimization of component and domain dependencies. We first describe the IUCBRF capabilities and then illustrate how they can be leveraged for instruction.

At time of writing, IUCBRF has been provided by request to a few dozen researchers and students, who have applied it to domains such as network fault analysis, visualization of case base properties, a calendar scheduling system, and a system for recommending codes for earthquake modeling (Aktas *et al.* 2004). In our own research we have applied the framework to several domains from the UCI repository (Blake & Merz 2000), the largest of these involving 20000 cases (the UCI *Letter* domain). While the code has not been heavily optimized, performance is adequate for our purposes. For example, on a Sun Blade 1000 (750 Mhz) the time to perform a simple kNN retrieval that scans an unstructured case base of 1729 cases averages approximately 0.15 seconds. Of course, performance also depends heavily on the particular algorithms used.

IUCBRF Infrastructure

IUCBRF provides a basic infrastructure for the implementation of components for the major steps in the CBR cycle, and for utilities to increase system functionality and usability.

Standard CBR Components

IUCBRF's implementation of the major components of the CBR cycle enables students to work with a full CBR system, potentially without understand the intricate details of each component. Instead, students are free to temporarily consider some components as "black boxes" (or perhaps "grey boxes"), so as not to be overwhelmed by full-system complexities while studying a few components. When the instructor deems it appropriate for course goals, students may return to these black boxes for a closer look. Instructors may also remove components for students to implement, or provide students with additional pre-implemented components, to allow for a gentle introduction and to help familiarize students with entire systems before in-depth study.

Utilities

In addition to CBR components, IUCBRF provides several utilities to assist in basic functionality and experimentation. Such utilities can play a crucial role in building a usable CBR system or studying its performance, yet would generally fall out of the scope of a CBR unit of an AI course. The utilities include:

- *Random generation facilities*, according to uniform, normal, and binomial distributions, applied to generation of domains, problems, cases, and weights.
- *Clustering facilities*, with k-medoid clustering already implemented, and an open framework for modifying and implementing other clustering approaches to use within the CBR system (e.g., for selecting prototypical cases).
- *Prebuilt GUIs* for all major components to assist in quickly building usable interactive systems. In much the same way that Java provides Swing components to handle the details of GUI widgets, IUCBRF provides widgets for CBR-specific tasks such as viewing cases, inputting problem information, and viewing basic component implementation details.
- *Performance monitoring* of the system, tracking measures including average retrieval and adaptation time, system competence, average solution quality, and a basic history.
- *Reference solution facilities* to provide an alternative means of solving a problem, for comparison to CBR performance or to treat the alternative method as an "oracle".
- *Support for experimentation*, according to leave-one-out and random problem generation strategies. Results are tracked by a performance monitor.

With the above utilities, students may focus on the CBR issues of interest and still obtain experimental results and a usable system without tedious implementation taking time away from the true course focus.

Minimizing Component Dependencies In IUCBRF

IUCBRF follows the design philosophy of independence of components, to allow the swapping of one implementation for another without requiring changes to other components. This is especially useful for performing comparative studies of component implementations. This genericity is accomplished through polymorphism—the ability of an object to have a standard interface, yet take one of any number of actual forms. The other components depend only on the interface, not the implementation details. To illustrate how this occurs, consider the example of features in problem descriptions. A problem consists primarily of a collection of features. The `Feature` interface includes basic operations such as a feature comparing itself to another feature of the same type. The collection of features in a problem may be implemented in any way, provided that an iterator on those features can be created. Thus the retrieval component merely depends on the interfaces of the feature types (for individual feature comparison) and the collection (for iteration). Details such as feature type, storage, and comparison need not be considered by the retrieval component, and thus can be changed without a ripple effect of changes across the framework.

Minimizing Domain Dependencies In IUCBRF

IUCBRF also follows the design philosophy of minimizing dependence on the domain, to allow a single system to be applied to several domains without requiring reworking or reimplementation of components. To illustrate, consider the problem feature example of the previous section. The similarity component does not need to know what features exist to compute similarity. It simply must be able to iterate through the collections of features and ask each pair of features for their individual feature comparison value. Thus the similarity component can be independent of the domain details, as well as the implementations of other components.

On rare occasions, some dependencies between a component and the domain are unavoidable. The primary example of this in IUCBRF is in the case adaptation component. IUCBRF does provide some standard domain-independent adaptation techniques, but some systems' adaptation components are inherently specialized with knowledge for their particular domain. However, IUCBRF is designed to handle even this situation gracefully, to minimize effort required to move a system to a new domain. This is accomplished via the *template* design pattern, in which domain-independent classes pass only a few domain-dependent operations to subclass implementations of abstract methods.

Thus by explicit representation of the domain, interaction through a standard interface, and deferring a minimal number of operations to domain-dependent subclasses, IUCBRF components remain independent from the domain. This means that a system built using IUCBRF to work in one domain can easily be modified for another, again allowing students and instructors to focus on the topics of interest.

Sample Exercises

To illustrate the educational use of IUCBRF, we present sample exercises applying IUCBRF to the study of key concepts in CBR. These exercises illustrate the advantages realized from independence between components, independence from the domain, and pre-implementation of CBR components and utilities. Note that these examples do not depend on a chosen domain, facilitating generalization of the lessons learned.

The Curse Of Dimensionality

In this exercise, students observe the consequences of irrelevant features in the problem representation of a CBR system.

- *Tasks*: Implement a domain from the UCI repository (Blake & Merz 2000). Then, for nearest-neighbor retrieval, run a baseline experiment using IUCBRF's leave-one-out capability, recording results with the performance monitor. For a few specific problem descriptions, examine which cases receive the highest similarity.

  Modify the domain definition to include two additional features, filled in with random values using IUCBRF's random distribution classes. (Thus, the features are irrelevant). As before, perform a leave-one-out experiment. Compare results and compare the retrieved cases for the same sample problem descriptions. Repeat this for 5 additional features added to the domain definition, and then for 20.
- *Questions*: What effect do irrelevant features have on system performance, and why?
- *Observations*: Students should observe that the addition of features increases retrieval times and decreases accuracy, and that the sample cases selected with no irrelevant features (the "true" nearest cases) tend to be buried by other cases that are near according to irrelevant features.
- *Learned Concepts*: Students will have seen first-hand the "curse of dimensionality"—that nearest-neighbor retrieval approaches can be mislead by additional irrelevant features. In addition, due to the heterogeneity of retrieved cases and the true nearest cases being "buried," classification by majority vote may be less decisive and accurate.

The Swamping Utility Problem

In this exercise, students examine the effects of increases in case base size on performance of CBR systems. The instructor should choose a domain from the UCI repository with sufficient cases to exceed the minimum required coverage of the problem space — such domains are likely to have few classes and many cases.

- *Tasks*: Implement the domain chosen by the instructor. Construct a system with a case library of 10 cases randomly selected from the domain. Perform a leave-one-out test using the performance monitor to track the system's performance. Repeat three times for different sets of 10 cases and compare results to assess expected performance with a 10-case case library.

  Repeat a few more times with higher numbers of cases, until you notice trends in system performance as case base size grows. Be sure to do enough experiments to establish that the trends truly exist and that you have fully captured their shape, and to consider the range of performance statistics.

- *Questions*: How is performance affected by increasing the numbers of cases? What trade-offs exist? Are there diminishing returns? How do you explain these results?
- *Observations*: Though results will vary based on the domain selected, in general students will find that for few cases in the case base, retrieval time is very fast, but solution quality suffers. For many cases in the case base, retrieval time is slower, but solution quality improves. However, as the number of cases continues to increase the rate of increase of solution quality is likely to level out while retrieval time will continue to increase.
- *Learned Concepts*: Students will learn the delicate balance between having enough case knowledge and having more than is needed, unnecessarily slowing retrieval. In comparing results with students working in other domains with other systems, students will see that this balance depends on the domain and system components used as well as the number of cases.

Case Base Maintenance

In this exercise, students compare the effects of three simple case base maintenance approaches on performance of system components.

- *Tasks*: Consider the following case base maintenance techniques already implemented in IUCBRF:
  - Null Maintenance - A new case is always added, and no case is ever removed, from the case base.
  - Periodic age-based deletion - A new case is always added, but cases can be removed in periodic offline checks if they have not been used since the last check.

  Implement a third technique by extending IUCBRF's abstract `Maintenance` class:
  - Threshold-based addition - A case is added if the difference between it and the closest case in the case base surpasses a fixed threshold. Cases are never removed.

  Build a system to run on a UCI domain with at least 200 cases. Set up the system with each maintenance technique in turn, and use IUCBRF's random problem experimental facility to generate 100 test problems to run. Observe the results with the performance monitor.
- *Questions*: How does the performance of the system over time compare for the three techniques, and why?
- *Observations*: Students will find that for null maintenance, the case base becomes large. They may observe problematic effects similar to those of the swamping problem exercise above. For periodic age-based deletion, the case base size will gradually increase and then suddenly decrease when unused cases are purged offline. For threshold maintenance, the case base does not grow even temporarily as it does for periodic deletion, but system performance is slower due to the more costly decision to add a case. On average, accuracy is expected to increase with increased case base size.
- *Learned Concepts*: Students will learn the importance of performing any extensive maintenance off-line, and how maintenance policies can avoid the detrimental effects of unchecked case base growth.

General Principles Of System Design

IUCBRF can also be used as the basis for more open-ended explorations. Once graduate students are familiar with the uses and tradeoffs of various components, they can use IUCBRF to design a system to study a particular research question, either proposed by the instructor or of their own choosing. If the framework has already been used in the course, projects can build on the components already constructed, augmented with new ones. By exploiting component genericity and domain independence, students can implement and compare alternative components and domains to investigate questions such as:

- Representation: What feature types are most suitable for problem and solution representation? Should any aggregate types be used to combine various simple pieces of data into a summarized or alternative form? What kinds of processing will need to be done in the CBR cycle for particular domains, and what representations will facilitate that processing?
- Similarity: What are the key considerations for the similarity measure? When is the standard Euclidean distance sufficient? How should non-numeric attribute types be compared? How should missing attributes be handled? How can good attribute weightings be determined?
- Retrieval: How is the choice of retrieval technique influenced by the other components, such as problem representation, similarity, and case base storage?
- Adaptation: What kind of case adaptation is most appropriate for a given domain? For a given domain, is there specific domain information available that has not been placed in another knowledge container, or can be more easily placed in the adaptation container?

For any of these topics, the instructor can guide students in a literature search for basic approaches and comparisons. Examining stated weaknesses of past approaches, or inferring weaknesses on their own, may give students ideas for extensions or alternative approaches. These alternatives and the original methods can all be implemented and applied in turn to a single system due to IUCBRF's component independence. Using IUCBRF's experimentation utilities, students can test their approaches without needing to spend extra time implementing such facilities. Experimental results can be obtained for several domains in turn on a single system configuration, exploiting IUCBRF's domain independence. Thus using the framework should enable students to quickly reach the interesting issues and to produce a system that is more reusable for future research.

Related Work

CIspace (Conati *et al.* 1999) and AIxploratorium (Greiner & Schaeffer 2001) are collections of applets demonstrating several AI topics, designed to be used as visual demonstrations of algorithms. They can assist in learning differences in behavior between algorithms, and are excellent for illustrating AI algorithms to non-programmers or when time does not permit detailed study, but are limited in their customization of domains and behaviors, as well as their possibilities for learning implementation details.

A more technical library is found in the AIMA code repository (Russell & Norvig 2002) which presents pseudocode examples and code for various topics appropriate for an introductory AI course. These examples are designed to illustrate particular concepts, for particular test domains.

WEKA (Witten & Frank 2000) is a collection of domain-independent machine learning algorithms written in Java, much in the same spirit as IUCBRF. WEKA is commonly used by machine learning researchers and industrial scientists, as well as instructors. While WEKA does not include code for the development of CBR systems, some of the implemented techniques could be embedded in components of a CBR system, including one developed using IUCBRF.

Thus, with the exception of WEKA, all the above tools focus on higher-level presentations with limited intent for general applicability, while IUCBRF is intended for a range of levels of presentation (depending on the number of "black-boxes" the instructor employs), and designed for reusability and generality within the context of CBR.

Conclusion

Many course projects require considerable time due to detours into algorithms and infrastructure orthogonal to the true focus of the course. In AI courses, this time requirement is often addressed by examining greatly simplified domains and systems, or by even considering systems only in the abstract. This paper argues that rather than making these sacrifices, instructors can use frameworks such as IUCBRF to provide the basic infrastructure, freeing students to focus on the topic of interest. By maintaining independence between components and independence from the domain, systems developed with IUCBRF can be reapplied to new situations, further reducing the required orthogonal efforts. In addition, utilities including experimentation facilities and standard GUIs can increase the usability of the systems. As shown in both the general discussion and the presentation of sample exercises, use of IUCBRF can enable rapid exploration of central concepts. Without the use of IUCBRF, any of the assignments could easily be a several-week project; with it, we expect that they could be completed in a week or less. We plan to apply IUCBRF to teaching an intensive graduate CBR unit in Spring 2005.

References

Aamodt, A., and Plaza, E. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications* 7(1):39–52. http://www.iiia.csic.es/People/enric/AICom.pdf.

Aktas, M.; Pierce, M.; Fox, G.; and Leake, D. 2004. A web based conversational case-based recommender system for ontology aided metadata discovery. In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID 2004)*. IEEE Computer Society Press.

Barron, B. J. S.; Schwartz, D. L.; Vye, N. J.; Moore, A.; Petrosino, A.; Zech, L.; and Bransford, J. 1998. Doing with understanding: Lessons from research on problem- and project-based learning. *Journal Of The Learning Sciences* 7:271–311.

Blake, C., and Merz, C. 2000. UCI repository of machine learning databases.

Blumenfeld, P. C., and Soloway, E. 1991. Motivating project-based learning: Sustaining the doing, supporting the learning. *Educational Psychologist* 26(3,4):369–398.

Boaler, J. 1997. *Experiencing School Mathematics; Teaching styles, sex, and settings*. Buckingham, UK: Open University Press.

Bogaerts, S., and Leake, D. 2005. IUCBRF: A framework for rapid and modular CBR system+development. TR 608, Computer Science, Indiana University, Bloomington, IN.

Conati, C.; Gorniak, P.; Hoos, H.; Mackworth, A.; and Poole, D. 1999. Cispace: Tools for learning computational intelligence. Accessed October 22, 2004 at http://www.cs.ubc.ca/labs/lci/CIspace/.

Gamma, E.; Helm, R.; Johnson, R.; and Vlissides, J. 1995. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison Wesley.

Greiner, R., and Schaeffer, J. 2001. The aixploratorium: A vision for ai and the web. In *Proceedings of the IJCAI Workshop On Effective Interactive AI Resources*.

Johnson, R., and Foote, B. 1988. Designing reusable classes. *Journal of Object-Oriented Programming* 1(5):22–35.

Kolodner, J. 1993. *Case-Based Reasoning*. San Mateo, CA: Morgan Kaufmann.

Krajcik, J. S.; Blumenfeld, P. C.; Marx, R. W.; Bass, K. M.; Fredricks, J.; and Soloway, E. 1998. Inquiry in project-based science classrooms: Initial attempts by middle school students. *Journal of the Learning Sciences* 7:313–350.

Leake, D. 1996. CBR in context: The present and future. In Leake, D., ed., *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. Menlo Park, CA: AAAI. http://www.cs.indiana.edu/ leake/papers/a-96-01.html.

Leake, D. 2002. CBR/CIP microprograms archive. www.cs.indiana.edu/~leake/cbr/code/.

Marx, R. W.; Blumenfeld, P. C.; Krajcik, J. S.; and Soloway, E. 1997. Enacting project-based science: Challenges for practice and policy. *Elementary School Journal* 97:341–358.

Russell, S., and Norvig, P. 2002. Artifical intelligence: A modern approach. Accessed October 22, 2004 at http://aima.cs.berkeley.edu/.

Thomas, J. W. 2000. A review of research on project-based learning. Available on the World Wide Web at http://www.k12reform.org/foundation/pbl/research/.

Witten, I., and Frank, E. 2000. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. San Francisco: Morgan Kaufmann.