An Incremental Model for Developing Computer-based Learning Environments for Problem-based Learning

Lin Qiu and Christopher K. Riesbeck
Department of Computer Science, Northwestern University, Evanston, Illinois 60201 USA
{qiu, riesbeck}@cs.northwestern.edu

Abstract

Problem-based learning (PBL) is a pedagogical strategy that centers learning activities around the investigation and development of solutions to complex and ill-structured authentic problems. It requires additional support and resources for students and instructors to use it in schools. Computer-based interactive learning environments have been used to provide students authentic and supportive settings for PBL. These systems, however, require significant up-front development effort before they can be put into use. In this paper, we describe an incremental model that allows instructors to author the learning environment during real use. In our model, an instructor is part of the feedback loop, complementing system feedback and collecting materials to be incorporated into the system. By working within the system, the instructor can observe detailed traces of student learning activities in the system and provide individualized coaching and critiquing. We describe our experience in developing Corrosion Investigator, a web-based learning environment, as an exemplar of our model. We focus on how the system is designed to facilitate instructor involvement and support incremental authoring. We present empirical results showing materials collected through use and benefits to the students and teachers.

1. Introduction

With a high demand for changes to the current drill-and-practice methodology of instruction, problem-based learning (PBL) has become a popular new paradigm of teaching. PBL centers learning activities around the investigation and development of solutions to complex and ill-structured authentic problems [2]. Students acquire problem-solving skills and knowledge through self-directed learning with guidance and resources provided by the instructor. While PBL offers an effective vehicle to improve teaching and learning, a number of difficulties occur in creating or implementing it in schools [5]. For example, doing activities in solving realistic problems such as collecting test samples or running experiments can be expensive, time-consuming, and even dangerous. Activities such as free exploration and idea testing can be overwhelming for students. Instructors need to provide additional monitoring and coaching to ensure students achieve desired learning goals.

To address these problems, computer-based interactive learning environments have been developed to help delivering PBL, e.g., sickle cell counselor [1], and Alien Rescue [6]. They organize components such as simulation, scaffolding tools, and student portfolios into a challenge-based structure. They provide a safe and responsive setting with easy access to task information, individualized feedback and critiquing, just-in-time learning, and scaffolding for the learning process. Such self-sufficient software environments reduced the instructor's time and effort from preparing and facilitating learning activities. They are, however, difficult and expensive to build. In order to support purely computer-based accurate feedback, the vocabulary of operations and situations in the system has to be specified in advance so that rules can be written. Once deployed, students can only do what the system has been prepared to support. It is considerably harder for instructors, as non-programmers, to modify a computer application when they want to customize it for their courses.

How then do we develop systems that do not need significant upfront effort but are still capable of supporting PBL? In the following, we describe a development model that allows instructors to complement computer generated feedback and incrementally develop PBL learning environments during real use. We believe it presents a practical and middle-road approach for developing PBL learning environments.

2. Approach

In PBL there is usually no single right answer to the problem and many paths to the solutions. Students are encouraged to approach the problem with their own strategies. Such open-ended activity means that the resources needed to support PBL can be cross-disciplinary and diverse. From our observations and experience in developing PBL curricula, we found that the development of content for PBL is an incremental staged process. First, curriculum designers design a PBL curriculum by choosing target content and skills, creating a motivating and authentic problem, designing possible student activities, determining supporting resources, and developing an evaluation strategy. Then the design is put into practice with students. Though the design was created by designers using their best knowledge, unanticipated situations still occur during practice. By handling those situations, the instructor improves his/her understanding of how students approach the problem, what common mistakes students make, and what appropriate advice to give. Through such learning experiences in using the curriculum, the instructor continuously incorporates new materials into the curriculum. Finally, the curriculum contains enough materials to handle most common cases and is ready to be shared and used by other instructors.

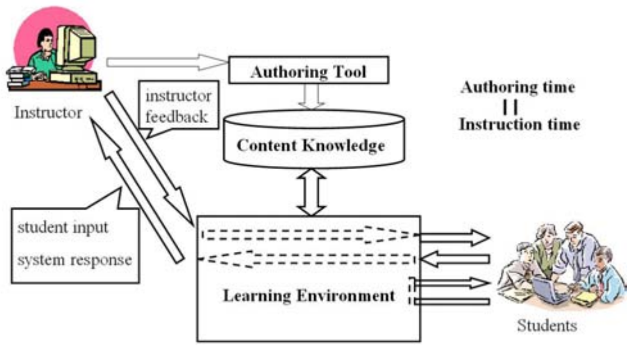


Figure 1. Incremental development model of computer-based learning environments.

Based on the above analysis, we propose a software development model (Fig. 1) that observes the natural incremental development process of PBL curriculum. In our model, an instructor works together with a system to provide support to PBL. While the system provides the primary learning environment with which students interact, the instructor complements the feedback from the system by helping to handle the situations that the system cannot. The instructor can also verify the feedback generated by the system before it gets to the students. More importantly, the instructor can introduce new materials into the system based on usage to incrementally improve the system's performance. Our model allows the instructor to have the benefits provided by computer-based environments, and at the same time lets the system be improved on demand to fit the needs of actual students. This supports content repair, refinement, adaptation and customization to diverse populations.

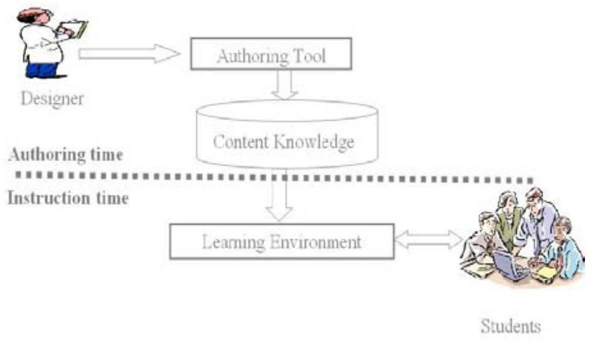


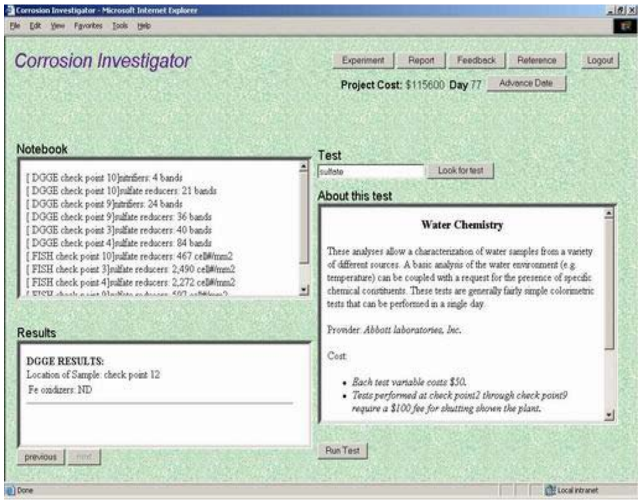
Figure 2. Traditional development model of computer-based learning environments.

Our model is different than the traditional development model shown in Figure 2. In the traditional way of system development, developers work with domain experts and educators to make sure the system has complete coverage of all possible actions and their appropriate feedback. This is not only hard during the authoring time, but may result in unnecessary effort spent on cases that rarely happen. More importantly, commonly occurring critical cases may fail to be collected. Furthermore, there is no easy way to add new operations, coaching, or critiquing during instruction time in order to incorporate new developments. Our approach is to let the instructor work with the learning environment and introduce new materials into the system. This allows authoring and instruction to happen at the same time and keeps the system from totally depending on pre-programmed content. There is no need to anticipate and implement all possible situations upfront. Instead, the system gradually migrates into a complete system through use. By having an instructor in the feedback loop, systems can be put into use during early development stages where automatic feedback generation is not yet mature and reliable. Issues not anticipated during system design can be explored and incorporated into the system during real use. This capability for customization can also keep systems up to date and usable after deployment.

In the following, we describe our experience in developing Corrosion Investigator, a PBL learning environment, as an exemplar of our model.

3. Corrosion Investigator

In the engineering domain, students need to learn not only engineering concepts but also the use of the conceptual knowledge as a set of tools. One important skill they need to learn is how to solve problems by running experiments and using the results to support or refute possible hypotheses. Corrosion Investigator (CI) is a computer-based learning environment for teaching environmental engineering students biological and engineering concepts in the domain of biofilms. In CI, students take the role of consultants helping a paper processing plant diagnose recurring pipe corrosion. Students need to decide which tests to run and which test results support the claims they make in order to solve the problem in a timely and economical manner. This requires students to fully understand the purposes of the tests and the implication of the test results.



[DGGE check point 10]nitrifiers: 4 bands
[DGGE check point 10]sulfate reducers: 21 bands
[DGGE check point 9]nitrifiers: 24 bands
[DGGE check point 9]sulfate reducers: 36 bands
[DGGE check point 3]sulfate reducers: 40 bands
[DGGE check point 4]sulfate reducers: 84 bands
[FISH check point 10]sulfate reducers: 467 cell#/mm2
[FISH check point 3]sulfate reducers: 2,490 cell#/mm2
[FISH check point 4]sulfate reducers: 2,272 cell#/mm2

Water Chemistry

These analyses allow a characterization of water samples from a variety of different sources. A basic analysis of the water environment (e.g. temperature) can be coupled with a request for the presence of specific chemical constituents. These tests are generally fairly simple colorimetric tests that can be performed in a single day.

Provider: *Abbott laboratories, Inc.*

Cost:

- *Each test variable costs $50.*
- *Tests performed at check point2 through check point9 require a $100 fee for shutting shown the plant.*

DGGE RESULTS:
Location of Sample: check point 12
Fe oxidizers: ND

Figure 3. The Experiment screen in CI.

CI is built based on the Indie Goal-based Scenario framework [4]. It consists of a set of interfaces: a welcoming screen showing the "challenge" document, a "reference" interface where students can browse materials describing the scenario and domain content, an "experiment" interface where students can order tests and collect results (Fig. 3), a "report" interface where students can construct arguments for and against possible diagnoses, using the evidence gathered from the tests along with explanations, and a "feedback" interface where students can read and respond to comments from their supervisor on their activities. More detailed description of CI can be found in [7,8].

CI has a number of features to support incremental development and allow interactions among the students, system, and instructor to be captured and incrementally incorporated into the system:

- The system provides an authoring tool to let instructors edit the content of the scenario with no programming. The authoring tool has a form-based web interface that allows the instructor to edit test specifications, test result generation, result display formats, background information, and so on. This lowers the barrier of content authoring and puts the instructor in charge during the incremental authoring process. It lets the instructor add and modify tests and explanations as necessary during use.
- The system's web-based architecture gives the instructor anytime, anywhere authoring access, and makes changes immediately available to students. All the content information is sitting on a central server. Both students and the instructor access the learning environment through a web browser. Changes made though the authoring tool update the content saved on the server and are immediately reflected in the learning environment. This web-based architecture avoids the difficulty of standalone software that has to be reinstalled on many computers.
- The system captures all student input, so that failures can guide authoring. For example, to be more authentic and encourage brainstorming, CI does not provide a menu of available tests. Instead, students specify tests by entering test names into a textbox. The system uses simple keyword matching to retrieve close matches. All student input is recorded and reviewed by the instructor. Unexpected but reasonable names can be added into the system using the authoring tool immediately. This also leads to discovering tests or test options that should be added to the system.
- The system puts the instructor in the loop. In particular, CI includes email links where students can contact characters in the scenario: the plant foreman, the plant manager, the scientific consultant and the supervisor. These emails are actually answered by the instructor. This approach to instructor participation provides a means to complement the system functionality, e.g., providing extra background information and test results, or coaching and advising, while observing the fidelity and integrity of the problem context.

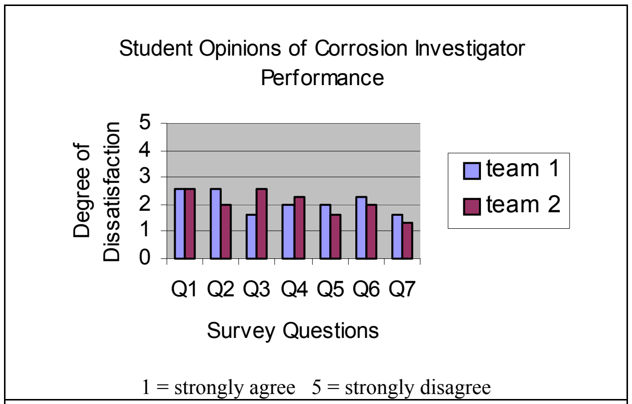
- The system captures instructor feedback. Specifically, CI organizes and displays student activities in the system (including tests that students have run, reasons for running those tests, claims and supporting evidence created, and the time and money that have been spent) into an interactive report form designed for easy critiquing. The instructor, in the role of supervisor in the scenario, can click on any item in the report and add comments to it. Students see and respond to these comments in the "feedback" interface. These critiques, based on actual student actions, are stored in the system. By adding patterns to these critiques, the system can suggest these critiques on future reports. Allowing instructors to work alongside the students providing ongoing formative coaching is considered an effective approach to foster learning in the cognitive apprenticeship model [3].

Working as middleware between the students and instructor, CI provides an authentic and scaffolded learning environment for students to perform PBL while alleviating the instructor from the work of data generation, facilitating the instructor in real-time assessment, and helping the instructor in accumulating the learning content in the system.

4. Results

In May of 2002, Corrosion Investigator was used in a class by six first-year graduate students in the Civil Engineering Department at Northwestern University. They were asked to form into two groups of three each, which resulted in a 3-male group and a 3-female group. Students completed a survey on their experience at the end.

According to the students' responses shown in Figure 4, overall the system was satisfying for doing the project. According to the professor, who created the scenario and delivered it previously without the software, the software significantly reduced his workload during the project phase. The workload was reduced from 24 man-hours to 4 man-hours. Meanwhile, students benefited from the immediate feedback generated by the system. The absence of delay reduced the project time from 8 weeks (which is the length when all the test results were generated manually by the professor) to 3 weeks. According to the professor, the quality of the student final reports stayed the same whether the students used the software or not. Both the students and the instructor agreed that the software should be used in next year's class. Though the data was based on six participants, it is still encouraging to see that Corrosion Investigator was capable in facilitating PBL.



Q1: I would like to use the system to construct a report rather than write it all by myself.
Q2: Overall, the interface makes me feel comfortable.
Q3: The system has provided enough support for doing the project.
Q4: Overall, this is an excellent system for doing the project.
Q5: Overall, the project has been completed successfully.
Q6: I prefer to use this system to run the tests and get the results back, instead of doing that via email with a person.
Q7: I would recommend this system to be used in next year's class.

Figure 4. Student opinions of Corrosion Investigator performance.

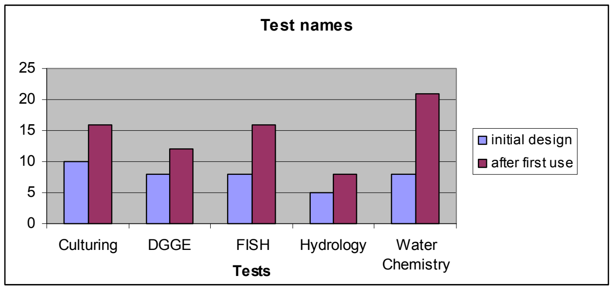


Figure 5. Test names for different tests in Corrosion Investigator.

When CI was first created, there were 39 test names in the system. During the first use, 34 new names were discovered through student input and added to the system, which resulted in a total of 87% increase of test names in the system. This significantly improved the system's coverage of student test inquiries. Figure 5 shows the increase of test names for each test. In addition, one student input helped us to realize that test variable "Fe" was missed from the Water Chemistry test. This test option and its corresponding test results were then added into the system using the authoring tool. CI's web-based architecture and authoring interface facilitated the improvement of the learning environment without interrupting the on-going project.

Table 1. Sample critiques on student work.

Student work	Critique
Test Result: [Water Chemistry check point 9]SO4: 83.08 mg/L Reason: High sulfate is still present, indicating SRB's may be active.	This is NOT evidence supporting chemical corrosion as a cause.
Test Result: [Water Chemistry check point 3]pH: 6.378 Reason: Neutral pH, indicating process is probably not a chemical one	There are other possibilities for chemical corrosion at neutral pH mn's- should acknowledge this.
Test Result: [Water Chemistry check point 3]H2S: 42.204 mg/L Reason: Rotton egg like odor indicative of sulfate reduction. High H2S concentration is indicative of SRB populations	That is correct- H2S a byproduct of SRB metabolism.

Table 1 shows some sample critiques from the instructor on a student generated claim and supporting evidence with test results and explanation. We analyzed the critiques and categorized them into three types. One type confirmed the correctness of the student work. A second type pointed out that the work is wrong, e.g., the use of a piece of evidence did not support the claim, or an ordered test was not necessary. The third type asked for more proof or explanation to justify the action, e.g., comparing the test results to other sites, or excluding other possibilities. Most of the critiques were considered reusable when reviewed by another domain expert. These critiques give us a basis for adding critiquing assistance in CI. We have been developing a scheme that uses the instructor's previous critiquing record and the current context to suggest likely critiques to the instructor. The instructor will be able to select or modify appropriate critiques to send to the students. This can save the instructor from writing the same critiques over and over again, but more importantly, it can prompt for overlooked situations and reduce the chance of missing critiques.

5. Discussion

What is the difference between pilot testing and the development process in our model? Pilot testing is usually done before the real use of a system. Its purpose is to make sure that the system has enough knowledge to function on its own when it is put into use. In contrast, systems in our model may never become completely autonomous. They may always need an instructor in the feedback loop to complement the limitation of the computer's ability such as natural language understanding. However, the instructor will be doing less and less work as the system becomes more and more competent to carry on work such as result generation, scaffolding, candidate critique selection.

Our approach is similar to the Wizard of Oz approach [9] in system development. Our model has a wizard working behind the system, but not for collecting data at a separate prototyping stage. In our model, such a process happens during the use of the system. Furthermore, the instructor does not pretend to be a computer, but works with the computer to complement the system.

To support open-ended learning, we believe the development of computer-based learning environments for such learning should also be open. Systems in our model facilitate incremental development by (1) starting with a sound educational framework (2) allowing early deployment and testing through instructor involvement (3) providing an architecture and interface for in-use authoring. We believe developing systems in such a manner provides a promising means for facilitating the development and customization of computer-based learning environments for PBL.

6. References

[1] Bell, B. L., Bareiss, R., & Beckwith, R. (1994). Sickle cell counselor: A prototype goal-based scenario for instruction in a museum environment. *Journal of the Learning Sciences*, 3, 347-386

[2] Boud, D, and Feletti, G. (1991). *The Challenge of Problem-based learning*. London: Kogan Page.

[3] Collins, A., Brown, J.S., & Newman, S. (1989). Cognitive Apprenticeship: Teaching the Craft of Reading, Writing, and Mathematics, In L.B. Resnick (Ed.) *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser*, Lawrence Erlbaum Associates, Hillsdale, NJ.

[4] Dobson, W.D. (1998). Authoring Tools for Investigate and Decide Learning Environments. *Ph.D. thesis*.

[5] Hoffman, B., and Ritchie, D. (1997). Using multimedia to overcome the problems with problem based learning. *Instructional Science*, 25: 2, 97-115.

[6] Liu, M, Williams, D., & Pedersen, S. (2002). Alien Rescue: A Problem-Based Hypermedia Learning Environment for Middle School Science. *Journal of Educational Technology Systems*, 30(3).

[7] Qiu, L., Riesbeck, C. K., and Parsek, M. R. (2003). The Design and Implementation of an Engine and Authoring Tool for Web-based Learn-by-doing Environments. *Proceedings of ED-MEDIA*, Hawaii, June 2003.

[8] Qiu, L., Riesbeck, C.K. (2004) Building Web-based Interactive Learning Environments to Facilitate Delivering Problem-based Learning. To appear at the *Annual Conference of the American Educational Researchers Association (AERA)*, 2004

[9] Wilson, J., and Rosenberg, D. (1988). Rapid prototyping for user interface design. In *Handbook of Human-Computer Interaction*, M. Helander ed., New York, North-Holland.