2010

Facilitating Teaching And Research On Open Ended Problem Solving Through The Development Of A Dynamic Computer Tool

Matthew Verleger
Purdue University

Heidi Diefes-Dux
Purdue University - Main Campus, hdiefes@purdue.edu

AC 2010-1815: FACILITATING TEACHING AND RESEARCH ON OPEN-ENDED PROBLEM SOLVING THROUGH THE DEVELOPMENT OF A DYNAMIC COMPUTER TOOL

Matthew Verleger, Purdue University

Heidi Diefes-Dux, Purdue University

Facilitating Teaching and Research on Open-Ended Problem Solving Through the Development of a Dynamic Computer Tool

Abstract

Model Eliciting Activities (MEAs) are realistic open-ended problems set in engineering contexts; student teams draw on their diverse experiences both in and out of the classroom to develop a mathematical model explicated in a memo to the client. These activities have been implemented in a required first-year engineering course with enrollments of as many as 1700 students in a given semester. The earliest MEA implementations had student teams write a single solution to a problem in the form of a memo to the client and receive feedback from their TA. For research purposes, a simple static online submission form, a static feedback form, and a single database table were quickly developed. Over time, research revealed that students need multiple feedback, revision, and reflection points to address misconceptions and achieve high quality solutions. As a result, the toolset has been expanded, patched, and re-patched multiple developers to increase both the functionality and the security of the system. Because the class is so large and the implementation sequence involved is not trivial, the technology has become a necessary to successfully manage the implementation of MEAs in the course. The resulting system has become a kluge of bloated inflexible code that now requires a part time graduate student to manage the deployment of 2-4 MEAs per semester. New functions are desired but are either not compatible or are too cumbersome to implement under the existing architecture. Based on this, a new system is currently being developed to allow for greater flexibility, easier expandability, and expanded functionality. The largest feature-set being developed for the new system are the administrative tools to ease the deployment process. Other features being planned are the ability to have students upload files and images as part of their solution. This paper will describe the history of the MEA Learning System (MEALS) and the lessons learned about developing custom teaching and research software, and will explore how the development of custom software tools can be used to facilitate the dual roles of teaching and educational research.

Introduction

Since 2002, Purdue University's first-year engineering problem solving and computer tools course (ENGR 106 – later renamed ENGR 126) has had a fall enrollment of between 1200 and 1700 students (350-800 in the spring). Students typically have their work evaluated by one of the up to 20 graduate teaching assistants (TAs), receive lectures from one of up to six faculty members, and potentially interact with one of the two full-time lab coordinators tasked with managing the day-to-day course activities. The size of the course is not trivial, and the effort required to implement any component of the curriculum is time consuming.

One of the largest pieces of that curriculum are Model-Eliciting Activities (MEAs). MEAs are realistic, open-ended, client-driven problems that require teams of students to develop a memo to the client describing a procedure for solving the problem. When MEAs were first implemented in 2002, the goal was to investigate improving gender equity through small group mathematical modeling[1]. To facilitate their work during the initial research, students developed a single draft of their procedure in lab and submitted it to a WebCT discussion board[2]. The success of that

research led to the continued use and study of MEAs in ENGR 106. In constructing a research program around MEAs, it was quickly determined that a custom software package would need to be developed to ensure that the necessary data was being collected in a format that could be more easily examined.

Developing custom educational software can be a difficult and time-consuming endeavor, but that software can also create amazing opportunities to investigate complex research questions over long periods of time. The purpose of this paper is to describe the evolution of this custom software package (now called the Model-Eliciting Activity Learning System – MEALS) and some of the practical lessons that have been learned about developing this software for use as both a research and teaching tool.

MEALS History

In 2002, when MEAs were first being implemented in ENGR 106, it was decided to eliminate face-to-face team interactions in an effort to better emulate the "real-world" team experience. To facilitate the electronic communication while still capturing the interaction and with MEAs consisting of only a single draft, the discussion board tool built into WebCT (the course management software used for the course) was selected to handle the process. Though the discussion board approach was functional for capturing simple interactions, setting up individual discussion boards for each team was extremely time consuming, the tool was generally inflexible for conducting more complex interactions, and the necessary linkage to WebCT reduced the long-term ability to store the data.

The Fall 2003 semester saw two critical changes. First was the decision, based on feedback from the students, to allow face-to-face interactions. The second change was to move from the WebCT discussion board to a custom in-house developed interface. An MEA consisted of only a single draft and assessment was based on the TA's perception of the students effort and completion, so the tool only needed to consist of a couple of mostly static web forms to collect the team's response, a script to process that data, and a single database table to store all the data. The simplicity of the tool meant that there was minimal need for any administrative tools; deploying a new MEA was as simple as creating a new database table and updating a variable to point to that new table.

In 2004, it was decided to focus the assessment of MEAs on the quality of the product being created instead of just perceived effort. A simple form containing a single open-ended textbox was created for the TAs to complete (with grades being entered directly into WebCT). Their feedback were added to the same database table as the student responses, and a simple dynamic page was created to show that feedback to the students. This sequence is shown in Figure 1.

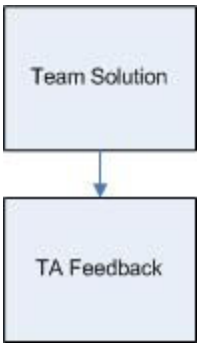

Figure 1 – Fall 2003 MEA Sequence

This approach was functional for two years before the research on MEAs necessitated a major redesign. By this point, MEAs has expanded from a single draft to a three iteration process, with the first and third iterations being evaluated by the TA and the second iteration being part of a double-blind peer review. Additionally, teams provided feedback to their peer reviews in the form of a short survey (termed the "Team Critique of MEA Critique by Peers"). Though the sequence had expanded, the contents of a given stage were still largely the same. The sequence is shown in Figure 2. The number of stages and the inclusion of both individual and team tasks necessitated moving from a single database table to multiple tables.

In 2006, a major overhaul was made to the rubric, necessitating another database redesign. Instead of only a single large textbox for both feedback and score explanation, four Likert items were used to numerically evaluate the responses and a large textbox was used to provide guiding feedback.

Over the course of the next year, deficiencies in how the rubric was measuring conceptual understanding were investigated, necessitating another rubric redesign with new unverified questions. Instead of updating the static form, the tool was patched to have the question stems to come from a database table. Because those questions were not going to be changed overly frequently (2 to 3 times per year at most), it was decided that those changes could be made manually in the database and no administrative tools were needed.

In the spring of 2007, a new developer (the first author) took over tool maintenance. The first task that developer undertook was to evaluate the current state of the tool, identify some of the future goals of the MEA research, and identify what functionality was not currently implemented that would be needed to achieve those goals. After evaluating the current state of the tools, it was decided that another major overhaul of the tool was needed. A new system was designed that was highly database driven, resulting in a highly adaptable system, but that was still built around the existing login authentication system. One of the key functions added to the new system was the ability to calibrate students for peer review.

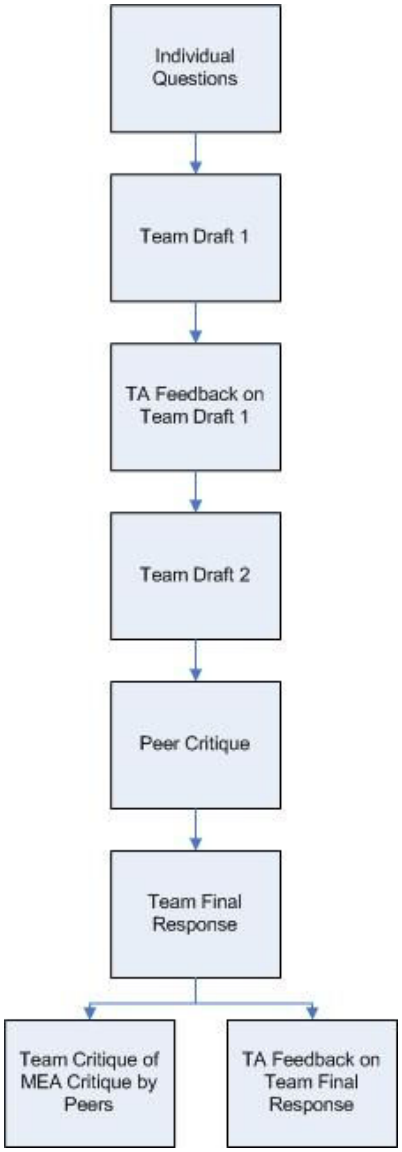

Figure 2 - Fall 2005 MEA Sequence

While the new system was highly adaptable, it was also sufficiently complex to the point that even the administrative tools that were created were not overly intuitive. Furthermore, the tools that were created were not fully capable of creating a new MEA, meaning that for each MEA, the developer had to manually update portions of the database. Over time, more administrative methods were created, but never to the point where a non-technical person could deploy an MEA.

The flexibility of the system allowed for advances in the research, particularly with the added ability to now calibrate peer reviewers and to freely re-order and alter stages. In Fall 2009, calibration and peer review were moved to the first draft, allowing students the opportunity to peer review generally lower quality solutions, effectively providing them more issues for which they are capable of providing feedback.

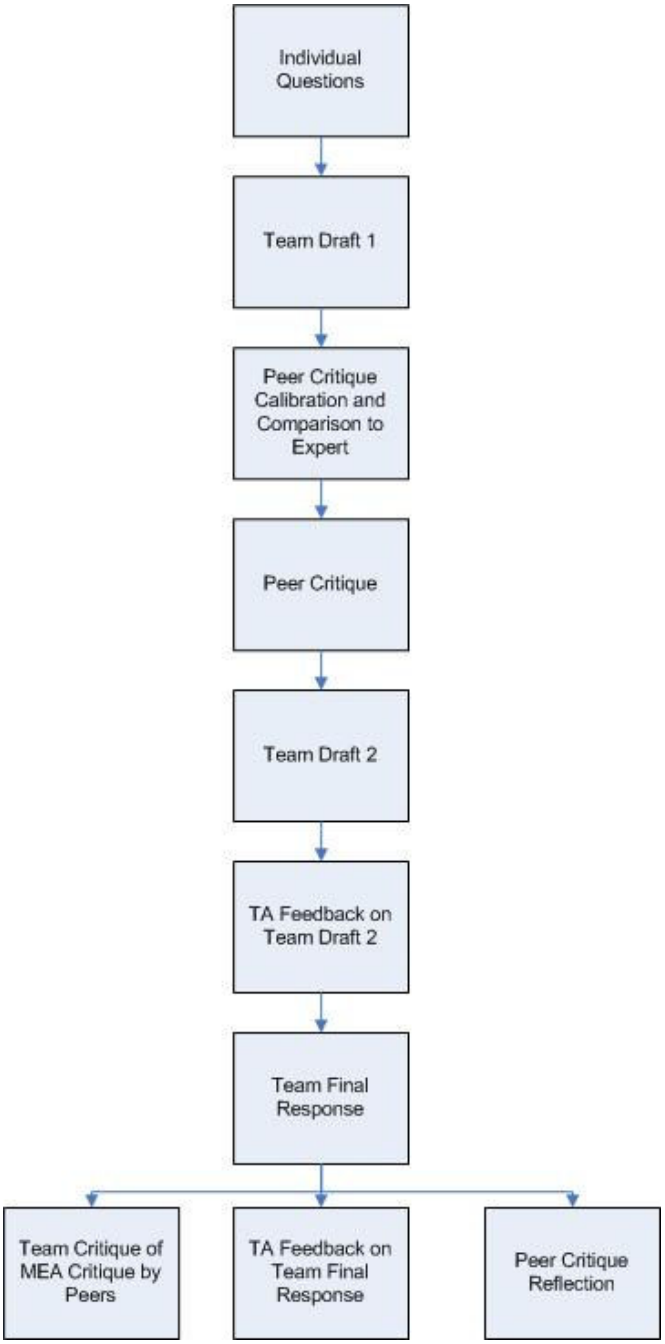

Figure 3 – Fall 2009 MEA Sequence

This new sequence can be seen in Figure 3. The new toolset, despite being more flexible, still lacked some of the basic functionality desired, specifically the ability to have students upload files and images. Because the system was tied to the university's authentication system and hosted by the university's IT program, the security of the system limited the ability to have efficient long-term storage of files. In 2009, the authors obtained a small digital content development grant which is being used to break free of the university's system. Currently under development, the MEALS system is designed to overcome these final limitations.

Lessons Learned

What follows are some of the practical lessons and advice about developing custom software tools for educational research that have been learned throughout the development life of the MEA tools.

Recognize the future evolution of your research
Research questions rarely occur as isolated events. In addressing one question, undoubtedly two more will present themselves. While you obviously can't anticipate the answers to all of those questions, trying to anticipate the pipedream goals of the research and what would need to happen to achieve those goals helps in planning your application. With the MEA tools, had the idea of altering the rubric or re-ordering of events been thought of sooner, the developers would have been able to preemptively structure the toolset to allow for this instead of having to implement dynamic behaviors in a statically designed system.

Decide how much interaction you want with your university's IT program
The MEA system has been tied to the university's authentication system. This made it easier for students to login, as it was a login and password they were used to using for other university tasks, but it reduced the ability to have non-university personnel access the system. This functionality was eventually patched in, but like most patches, it never functioned as organically as was desired. Likewise, being tied to the university's authentication system also required the use of the university's hosting resources, with all the benefits and limitations that brought.

Select a development environment you can control
If you choose to break away from your university's IT system, select a development environment that maximizes your control. Server space and domain names can be purchased for as little as $50 per year, so cost is a trivial issue in the process. Don't feel tied to an uncomfortable or inflexible architecture just because it is what your university makes available to you.

The majority of the MEA tool implementations were created under the Zope development environment. Zope security layer prevents functionality that is common to other environments and as a result hindered the ability to easily implement some basic functions such as file uploads. The MEALS system is being developed as a PHP/MySQL, where such functionality is native.

Avoid arbitrary limits unless absolutely necessary
Oracle's VARCHAR2 (variable length character string) data type can hold a maximum of 4,000 characters, while the CLOB (character large object) data type can hold a maximum of 4,294,967,296 characters. When the first MEAs were being completed, students were writing 1000-2000 characters, making a VARCHAR2 column more than capable of storing the data, but as more iterations occurred and more feedback was being given to help students improve their solutions, the length of solutions began to grow. The 4000 character ceiling became a major hindrance. Students had more to say, but the system simply had no means of storing that data. The limitation of having selected VARCHAR2 storage methods instead of the slightly more complex CLOB methods required a line-by-line review of the entire tool codebase.

Create administrative tools first

The lack of a proper administrative interface has been the single largest problem with all previous versions of the toolset. Because there is a "this is how we did it last time and it worked" mentality, if the administrative tools are not created first, they more than likely will not ever get created. For creating the newest MEALS system, the administrative tools are being created first and used to develop the student interface. This ensures that any functionality needed for the students is coming from a functional administrative tool and not a quick database hack.

User tracking
Track every step your users take. It is not uncommon to receive an e-mail from a student claiming to have completed the task and that surely "the system" must have lost their data. By documenting when a student logs in or loads a page, you are creating a more detailed documentation trail. User tracking can also be useful for informally estimating time-on-task, though it is not a scientifically accurate method. Because students can have a page open without being on task, the amount of time spent on a particular page is only an estimate of time-on-task, but this value can be useful for determining where students are spending an excessive amount of time trying to solve what you consider to be trivial problems.

Timestamp everything
By the time the data being collected has been analyzed and is being written up, you won't remember specifically when tasks occurred. Documenting timestamps on everything provides an automatic documentation of when actions occurred.

Don't delete data – flag items as inactive
Proper software development says that data should never get deleted, but this recommendation has more to do with good research technique. By specifying data as active or inactive, you will be able to reconstruct a more complete sequence of events as they occurred. Much like the recommendation to timestamp everything, it will also help in developing a write-up of the research if there is a guarantee that no data is missing.

Have a change management plan
Recognizing that projects evolve and interfaces and questions change throughout the life of a research project, having a change management plan in place to prepare for those changes becomes essential. One of the problems with the earliest versions of the MEA tools is that their static nature meant that changes were rarely properly documented. When a new question was added, the static form was simply updated without ever archiving the wording of the old question. Those changes are irretrievably lost. This becomes vitally important as the toolset becomes a longitudinal information system. Being able to recreate what a student saw at any time becomes important in tracking the evolution of the research.

Find the right developers
Finally, more important than planning the software is finding developers who can and want to invest themselves in the project. The developers for the first iterations of the MEA tools were all highly skilled programmers, but they were not invested in MEAs and the MEA research. Because they were not invested, they could not foresee potential areas where the MEA system would need to grow. They were limited to only what they were told. Likewise, the researchers didn't fully understand what would be possible to implement, giving them a limited ability to

describe what features they think would be useful. As the MEA research grew, the software was not prepared for that growth, and as the software reached the limits of its ability to grow, the research became limited in its ability to answer questions.

Closing Remarks

Building custom software for educational research is difficult, but it can also become a cornerstone of a research program, offering longitudinal data and an ability to adapt to a wide variety of circumstances and questions.

References

1. Zawojewski, J.S., et al., *A Modeling Perspective on Learning and Teaching Engineering Education*, in *Models and Modeling in Engineering Education: Designing Experiences for All Students*, J.S. Zawojewski, H. Diefes-Dux, and K. Bowman, Editors. 2008, Sense Publishers: Rotterdam, The Netherlands.
2. Diefes-Dux, H. and P.K. Imbrie, *Modeling Activities in a First-Year Engineering Course*, in *Models and Modeling in Engineering Education: Designing Experiences for All Students*, J.S. Zawojewski, H. Diefes-Dux, and K. Bowman, Editors. 2008, Sense Publishers: Rotterdam, The Netherlands.