The Advanced Team Project Course, Or How to Manage a Six-Ring Circus

Richard A. Brown
Department of Mathematics, Statistics, and Computer Science
St. Olaf College
rab@stolaf.edu

Abstract

Interest in undergraduate research has boomed in recent years. The traditional model of directing a student or two in independent research during a term, added gratis to a full load of teaching and other academic responsibilities, cannot scale upward to meet the demands of a greatly expanded role for research experiences at the undergraduate level. Therefore, as part of an expansion of our CS program, we have introduced a course CS 350, Advanced Team Project, in which students pursue multiple simultaneous undergraduate research projects within a regular, full-credit course. We report on the experience and lessons learned from this first offering of the course during January 2006.

Introduction

Interest in undergraduate research has boomed in recent years. Many see collaborative research projects as an effective way to place a disciplinary education in context and prepare students for the kind of activities they will likely encounter throughout their 21st century lives. Undergraduate research requires resources in order to thrive, measured not only in laboratory space, equipment, and supplies, but also in terms of human capital, particularly faculty supervision. The traditional model of directing a student or two in independent research during a term, added gratis to a full load of teaching and other academic responsibilities, cannot scale upward to meet the demands of a greatly expanded role for research experiences at the undergraduate level. Expecting more independent research projects is not only unfair to a professor, taking a toll on faculty energies, but also naturally tends to limit the scope and quality of student research experiences.

Therefore, as part of an expansion of our CS program, we have introduced a course CS 350, Advanced Team Project, in which students pursue multiple simultaneous undergraduate research projects within a regular, full-credit course. This not only makes it possible to assign teaching credit for research supervision, but also address several other needs of the program. Our experience with the first offering of CS 350 informs other institutions who may be contemplating a similar approach to recognizing and supporting undergraduate research at a liberal-arts institution.

Institutional context

St. Olaf College is a liberal-arts institution with some 2800 students located in Northfield, MN. The Computer Science Program has fortunately been housed in the noted Department of Mathematics since its inception in 1975. The Program offered a six-course Concentration for nearly three decades before expanding to a CS Major in 2002.

The College has long valued undergraduate research, and the natural science disciplines have excelled in this regard. With the help of grant support, over two dozen students have pursued summer scientific research in each of the past several years under local faculty direction. Participation in summer undergraduate research, though unpaid, is explicitly expected of new science faculty members. However, relatively few members of the (recently renamed) Department of Mathematics, Statistics, and Computer Science(MSCS) have seldom directed summer students until recent years. Certain MSCS faculty have offered independent research to numerous students, a course for which students but not faculty receive credit.

Undergraduate research has blossomed in the computer science program over the past decade. Most of our research projects have involved collaborations among several students, and the research problems have often had an interdisciplinary aspect. For example, in one project carried out in Fall 2000, a team of four students created *Olafractal*, software for experimenting with fractal images generated from linear transformations. The problem was suggested by a Mathematics professor and continues to be used as a teaching tool in Linear Algebra and upper-level Geometry courses. The team designed and implemented the work largely on their own (their sessions in the CS lab inspired many underclass students in the lab working on course assignments), and they presented their results in a colloquium talk and also in a poster session at a local science symposium (Barnard et al., 2004), (Eliason et al., 2001a), (Eliason et al., 2001b).

Not only is the research experience good for students, but they like it. Thus, research was at the forefront of our thinking as we engineered the new CS major curriculum, doubling the number of CS course offerings.

Our current major represents preparation for and execution of undergraduate research at every curricular level.

- *Foundation courses.* The course *Software Design and Implementation* (CS 251), which has CS1 as its sole prerequisite, includes a month-long team project based on a waterfall-model software lifecycle. This course is a prerequisite for most intermediate and advanced courses in the curriculum. Students emerge with a common experience of designing and executing a highly structured project, which gives them a shared basis for subsequent individual and especially collaborative projects. Even though they seldom adhere to this strict development model in later work, they all know how to carry out a team research project, even before they start to do any real research. For example, this common experience enabled the Olafractal team (Barnard et al., 2004) to organize itself and to complete its work within a single term, in fact, ahead of schedule.

  Recently our CS1 course, *Principles of Computer Science* (CS 121), has incorporated multimedia applications using a wiki whose authoring language is Scheme. This experimental endeavor has given CS1 students an opportunity to participate in creating the new system, building what nobody has built before in it, thus tasting a research-like activity in the introductory course.

- *Core (intermediate) courses.* Several of the core courses include research opportunities among their assignments. For example, *Operating Systems* (CS 273) includes an individual project to implement one or more new system calls to the Linux kernel that add some verifiably new capability to the operating system. Students are guided in how to add a new system call that is a clone of an existing call, then they must identify a new feature to add and research how to perform that modification, finally they must implement their modification. Although some books are helpful as references, students must also glean specific data structures and algorithms directly from the source code of the particular kernel version being modified, and often use web-based resources such as Google or topical sites, sometimes electronically consulting with off-campus persons in order to solve their problem.

Two project-based courses stand out for their relationship to undergraduate research. *Client-Server Applications* (CS 284) introduces the Java language (prerequisite courses present Scheme and C++) and SQL, and uses Java to explore graphics user interfaces (GUIs), network programming and client-server structures, and programming interfaces with XML and with database management systems. These topics are unified in a large-scale team project that is originated in the course, in which the team consists of all members of the class. The team uses an agile software development methodology (contrasting with prior waterfall-model experience), an approach that is new to these students. More experienced students complete the basic material at an accelerated rate in order to begin advance work on the project sooner; they must frequently carry out the same kind of research activities that CS 273 students must, consulting with web and other resources in order to solve non-standard design and technical problems. The course project is chosen by the professor to include new ideas, and with an actual clientele in mind.

For example, the Spring 2005 offering of CS 284 originated the Escher project, an ambitious multi-team effort to produce software that makes it convenient for novices to create and publish online web-based portfolios (Etshokin et al., 2006). Many of the design concepts of Escher are new among portfolio management systems, for example, *annotated links* that allow a user to add comments to a hyperlink that may represent an intellectual connection in the user's portfolio work. The software is being developed with a particular audience in mind, namely St. Olaf students pursuing individually designed majors. These students must each produce an online portfolio documenting their progress through their majors, although these users seldom have prior web-related skills. Work on Escher has led to several local research presentations and a MICS paper (Braasch, 2006).

Another course with an all-term research project is *Ethical Issues in Software Design* (CS 263). Our approach to this course recasts computing ethics from the analysis of policy to the learning of skills and knowledge that support the ethical practice of computing in organizations (where almost all computing happens). These principles are applied in a large project to perform an ethical analysis of a particular software system and its socio-technical environment. For example, two offerings of the course (Springs of 2004 and 2005) studied a new locally developed student information system now in use at the College. The present offering (Spring 2006) is examining the `northfield.org` community web site (Wiggly et al., 2006). Separate teams of three to five students are considering ethical and social issues inherent in the user interface, the overall editorial choices of layout and content, and a feature in which community members may post their blogs on the site. Specific questions (e.g., “Would posting political campaign blogs constitute an endorsement or campaign contribution?” or “Does the user interface favor one constituency over another?”) are examined using background research (e.g., examination of campaign laws) and social science methods (e.g., focus groups, surveys, talk-aloud protocols with the interface). This course provides valuable experience in a contrasting form of research in computing with considerable practical impact.

- *Advanced courses.*

  Besides CS 350, several advanced courses and electives have significant undergraduate research content. For example, *Bioinformatics* (CS 315) includes readings and presentations from the bioinformatics literature and applied projects to build new software tools with applications to biological research. Also, students the *Seminar in Computer Graphics* (CS 300) began work on a challenging project in 3D computer visualization, called the Correspondence Problem, using a new pipelined approach (Etshokin et al., 2005). Finally *Senior Capstone Seminar* (CS 390) is a research-oriented course taken by CS majors in their final year, in which students make contributions to an ongoing research project such as Escher, document that project, perform ethical analysis of it, present readings from the research literature, and write and present a research paper of their own.

The weakest prerequisite for CS 350 is CS 251, the foundation-level software design course that includes a waterfall-model team project. With these opportunities for research in other CS courses, plus summer research, most students taking CS 350 enter the course with at least some prior research experience.

Why offer CS 350 if students encounter undergraduate research without it? The purpose of the course is not to give every major a chance to participate in research—the Senior Capstone Seminar insures that. Instead, CS 350 provides a course home for students who want to carry out projects of their own that might not otherwise appear in the curriculum, and to continue projects that may have started in other courses or summer research. Without the course, students would have to petition faculty for Independent Research CS 398, for which students receive credit but faculty receive no teaching credit.

Indeed, the idea for CS 350 originated in a term (Spring 2001) in which the author was leading two separate projects under the CS 398 registration, totalling five students, while other CS faculty were also directing independent research. In all, nine students

project name	*languages*	*new?*	*team size*	*sr/jr/so*
Escher	Java,XML,SQL	ongoing	4	2/1/1
CPET	js,php	ongoing	3	1/1/1
Beowulf	C++,etc.	new	3	2/1/0
Softw Des	C++	new	3	1/2/0
SIS	proprietary	ongoing	1	0/1/0
WebEd	Java,js,elisp	ongoing	1	0/0/1

Figure 1: Summary of projects in January 2006.

were pursuing independent research under CS 398. The strategy of CS 350 is to consolidate most of those projects for multiple terms into a course, for which teaching credit could be assigned and in which interaction between teams could take place. The purpose of the "team" requirement in CS 350 is largely to make that course feasible for the professor, as well as to continue the program's emphasis on collaborative work.

Structure of the course

The first offering of CS 350, *Advanced Team Project*, took place during January Interim, 2006. This is a four-week block-style term in which students take a single intensive course. The Interim setting enabled teams to work for long sessions together, with the possibility of multiple meetings per day, and without the interference of other courses. The primary workplace was the CS computing laboratory (which was large enough to house all of the projects simultaneously). A regular classroom was available at scheduled times each day for meetings and presentations. Fourteen students participated in the initial offering.

The projects were chosen by the professor who had responsibility for managing them. Project proposals were invited from the students, and a list of eight new and existing projects was also provided, with descriptions and links to further material. One of the listed projects required only C++ programming skills at the level the prerequisite CS 251; others required additional experience, learning new languages and systems, etc. Students were encouraged to state any preferences for which project they might work on, in consultation with the professor. The professor made the final determination of projects and teams.

The six projects are summarized in Figure 1.

- The *Escher* project originated in an offering of CS 284, and is described above. Three of the four team members had previously worked on Escher; all had taken CS 284.
- The *CPET* project originated in summer research. Unlike the other projects, the work consisted of three relatively independent efforts. One student had worked previously on *CPET*. Another was a transfer student in a first team project experience, not having taken CS 251.
- The *Beowulf* project's goal was to create and document a Beowulf cluster from castoff machines, as a prototype in preparations for a subsequent acquisition. All three students had taken operating systems; one had considerable hardware experience, and another had two years of summer research experience.
- The *Software Design* team was charged with developing materials for CS 251. This was a two-part task, first adapting a suite of object programming exercises to a biological domain, then starting a graphical computer gaming application as an additional source of exercises for the course. This team had the weakest backgrounds among students in the course.
- The *SIS* project was part of the Computer Center's effort to create new student information system for the College, a comprehensive system for providing innovative integrated services to the Registrar, faculty, and students, and the subject of two terms of CS 263 ethical analyses. The student was part of team consisting of computer center personnel and user representatives, and was charged with developing recommendations and partially implementing a new section of the system. She had worked on the project for two years, including two summers.
- *WebEd* is a client-server application being developed to support the wiki-based CS1 course. The CS 350 student on the team worked with another student (senior) who was not taking CS 350, and with the professor. The CS 350 student on the team was also a member of the Escher team.

These projects represent a heterogeneous collection of new and continuing projects with diverse goals, languages, and settings, with students with wide-ranging backgrounds and experience levels. Most teams consisted entirely of team members, but two students were members of external teams.

Individuals were required to record or "log" their time on the project using a web-based form, including nature of each recorded activity, the starting and stopping time for each activity (as well as the timestamp of the log), who they collaborated with, etc., with a blank for recording any comments. Analysis of the resulting database was essential for assessing the individual contributions of each student, and the students knew this. The completeness of the information made it possible to take into account such issues as the delay (if any) between performing and recording the work. Daily and cumulative totals for each student were posted in an anonymous way, providing feedback on student progress and enabling students to compare their totals with their peers'.

Each team was initially charged with a two-day assignment to formulate a plan for their term, with target dates and deliverables, and to present those plans to the class. This assignment not only gave direction and tangible goals to each team, but also began a practice of teams presenting their progress to each other. All teams readily grasped what needed to be done for the assignment, and worked out satisfactory plans in consultation with the professor.

Presentation sessions were continued on a daily basis in the form of 5-10 minute brief updates. The four full teams were asked to give reports on nearly a daily basis; the two individuals gave only occasional updates. This provided a natural setting for demonstrations when milestones were reached. Also, the SIS student presented the proprietary system she did her work on, which gave the others exposure to an alternative development environment.

The course met five days a week. On a typical day, the first student or two would arrive in the lab at about 8:30 am and begin working on their projects, and the professor too would begin his work at a lab station. Others would gather throughout the morning; the lab would be full by 10:30 or 11, and buzzing with multiple conversations, a team meeting or two, and consultations with others about technical matters. These consultations were often with other team members, sometimes with the professor or with members of other teams who had worked on that project previously or had certain expertise. Occasionally some banter would break out among students; rarely, these eruptions would have to be tamed by the professor. Most students broke for lunch in twos or threes, coming and going throughout the hour or two before 1:00, when the daily meeting took place in the alternate meeting room (furnished with display equipment but not individual computers). The meeting would last 30 or 45 minutes, an hour if the professor had a lot of things to announce or if a team had a longer demo. Then the entire group would return to the lab for the afternoon, with individuals and teams eventually leaving from time to time. By 5:00, the professor and only a handful of students would remain, and the lab might be empty by 5:30.

The professor spent little time in his office, working instead at a lab station throughout the day on both course-related and other tasks. Sometimes he was calling or e-mailing computer center staff with a special IT need to support one of the teams. On some days, most of the day would be devoted to a single team's project, building or modifying some software component that would take the team too far afield. A significant amount of time went to other course-related projects, such as performing analysis of the logging database, or to giving feedback to teams on documentation. The professor was available to consult with teams at any time, and would convene a partial or full team meetings as needed, perhaps seeing one or two teams per day.

At the end of the term, each team (except WebEd, as explained below) made a final 15-20 minute presentation to which other interested parties were invited. CS faculty and upper level students were invited to all sessions; the SIS project director was present for the SIS presentation; representatives of the Center for Integrative Studies, which administers individually arranged majors, attended the Escher presentation; some natural scientists came to the Beowulf talk.

Two kinds of reports were required at term's end, both presented as web-based forms. Each team reported on their accomplishments, identified their deliverables and their locations, and indicated what needed to be done next. A separate form was developed for each team, and considerable attention was paid to making sure that a future researcher that is new to the project could get started readily, and that the supervising professor could find something that was needed without having to read large amounts of source code or documentation.

Each individual also completed an end-of-term form, giving the students a chance to identify their contributions and the ways that others (not necessarily on their own team) collaborated and helped. The form solicited comments about how well the team functioned, including both strengths and weaknesses, and asked students to assess their own relative contributions to the team's accomplishments. Responses on these forms often provided insight into the work each person did that could not be observed by the professor sitting in the lab.

Experiences from the first offering

The first offering of the course was a success. Almost every team made great strides on their projects and largely accomplished their goals for the month, and students were justifiably proud of their work.

- The Escher team made extensive improvements to the GUI, added several features, and made significant improvements in the internal structure of the code. One student researched and implemented a convenient and maintainable strategy for distributing the software to both users and developers. User documentation was begun.

- CPET is a client-server application in which a server with customizable capabilities is accessed through a browser plugin or a server-side application (Mueller et al., 2004). One CPET team member completed an extension of the browser plugin that automatically annotates HTML pages with reference links, an application suggested by a Chemistry professor. Another student created a server-side module for the course management system Moodle that enables customizable quiz questions, for which student responses (perhaps containing program code) are processed by a CPET server (perhaps evaluating that code) before storing the results in Moodle databases. The third team member helped research Moodle modules and other issues, and performed other configuration and maintenance tasks on the system.

- The Beowulf team succeeded in assembling and demonstrating their prototype cluster. They researched and implemented system management procedures for running the system and for adding nodes, selected protocols for distributing computations, and documented their procedures for future teams.
- The Software Design team completed their biology-oriented exercise suite on time, including external documentation, and made considerable progress on implementing a graphics-oriented computer game.
- The SIS student accomplished many of the term's goals, in spite of significant loss of project time due to problems when upgrading the proprietary software used by the project.

The course also succeeded in that the professor found it possible to manage the fourteen students involved in multiple unrelated team projects. This quelled any fears that a course such as CS 350 might simply not be feasible. The students' previous experience with research and team projects surely helped—it probably wouldn't have been possible for a single professor to *introduce* that many students to research and collaborative work with so many and such diverse projects. Teaching in the Interim was also a benefit for this course in many ways: students had no competing courses, and could work extended hours in a common location with full access to the professor; finding a shared meeting time was never a problem.

However, the course didn't work for the professor in one respect, in that he took on too many tasks on behalf of teams.

- The students on the CPET team did not have the expertise to make significant modifications of the CPET server, which is implemented in Java, Python, and SQL. Extensions needed to support the automatic annotation feature for the plugin required about three days work for the professor. None of the students knew the Python and SQL needed for the extensions, and the problem arose too late in the term for one of them to learn the necessary skills. Even if student implementation had been feasible, it would have displaced that student's other project responsibilities.
- A test suite was required for the series of biology-related exercises developed by the Software Design team. This suite needed to be especially complete, robust, and correct in order to serve as a teaching tool against which future

CS 251 students could test their implementations. This test suite was quick for the professor to generate, but the professor also had to implement the entire exercise suite to be tested, in order to test the test suite with an implementation he could have full confidence in. This took about 1.5 days of full-time work, spread out over three days.

- The WebEd project involves connecting an `emacs` editor to a text area in a web page (e.g., for the CS1 wiki) using a client-server architecture. The two students on the team were successful in completing the javascript browser code, Java communications applet, and the Java intermediate server, based on prior work. But the `emacs` interface is a sophisticated application in the emacs lisp language. Since one of the students team members was on the Escher team and the other one wasn't even a member of the class, the responsibility fell to the professor, who had written the original `emacs` interface code. This couldn't receive the same priority as the other five projects, being an add-on for the team member enrolled in the class, and the task arose too late in the term to be completed by the professor. Therefore, the WebEd project fell by the wayside.

These implementation commitments impeded other activities related to the course, e.g., timely feedback on external documentation drafts. Speedy feedback is crucial for the quality and success of a project-based Interim course. Thus, a professor teaching CS 350 must be very careful about the development commitments he or she makes to individual teams.

The daily presentation sessions were not well received by the students, who largely saw them as interruptions to their work. Their purpose was threefold: to develop students' skills in giving informal presentations of their work; to give teams feedback on their work; and to increase interactions between teams.

The quality of these presentations was often quite low. For a next offering, we will provide quality guidelines for these 5-minute talks, and include some assessment to make sure that the guidelines are taken seriously.

Few of the students provided feedback to other teams during the brief presentations; the faculty member asked most of the questions. Some students seemed reluctant to say anything that might be perceived as critical of another team. Others seemed mostly interested in making the session as short as possible. But this kind of peer feedback can greatly benefit a project and can sharpen the analytical thinking and communication skills of a questioner. Next time we will encourage it by tracking how often people ask questions, and how good their questions are.

In a group feedback session with the class near the end of the term, students indicated that they didn't object to the goals of the short presentations, but to their daily

	total reported		*reported within 2 days*	
student code	*total hours*	*mean hours*	*total hours*	*mean hours*
17	121:35	06:23	92:05	04:50
47	120:00	06:18	58:15	03:03
63	117:30	06:11	111:00	05:50
83	108:10	05:41	80:55	04:15
66	104:20	05:29	46:55	02:28
80	96:00	05:03	96:00	05:03
10	94:25	04:58	83:55	04:25
57	92:25	04:51	88:40	04:40
25	72:55	03:50	63:20	03:20
30	66:45	03:30	44:45	02:21
22	65:55	03:28	41:00	02:09
29	64:50	03:24	37:25	01:58
38	61:25	03:13	49:15	02:35
46	55:40	02:55	44:25	02:20

Figure 2: Hours logged on the project per student.

frequency. They felt that sessions three times per week, with each team presenting twice per week, would be sufficient. We will gladly trade frequency for quality if it will help to encourage the sense of a larger collaborating research community among the teams in the course.

The logging system proved to be invaluable for assessing student work in quantifiable ways. For example, Figure 2 shows the hours logged per student throughout the term. It shows that no student averaged as much as seven full hours throughout the full 19 days of the term. Our perception is that some students in prior Interim project courses have spent more time on their courses, but having no comparable data, we can't be sure. We might announce some expectations for these hours in a future offering in an attempt to increase these numbers. However, since we are generally satisfied with the accomplishments of student teams, there may be no need to set higher targets.

Nevertheless, we were disappointed with students who spent less than four hours per day on the course, since this was the students' only course. One student averaged less than three hours per day—little more than class meeting time alone would be for a minimal full load during a semester! The numbers provide objective evidence for our impressions of the commitment level for this and many other students.

The final two columns indicate hours that were reported within two days of the occurence of the work. In most cases, we consider this as a sign that the reported figures are actual values, not estimates. While some students were quite prompt about reporting their time, others were frequently late. In future offerings we plan to call for prompt submission of logs.

The team and individual forms were indispensable for evaluation of student work. Whereas the logs provide objective quantities, the team report enumerates tasks completed and records how to pick up the project later, and the individual report gives a student a chance to identify the nature and extent of their own contributions and provide subjective comments that lend insight into the team process. The combination of these items constitutes a broad array of varied criteria, providing plenty of information and substantiation for assigning a grade.

The projects whose teams were wholly class members worked well in CS 350, but the two projects with external team members were less satisfactory. As indicated above, the WebEd project stalled when there was insufficient time to complete a key component, to be implemented by the professor. A comparable project without such a dependency may well have reached its goal; also, WebEd didn't merit the same priority within the course, since the class's student participant in WebEd was also on another team.

The SIS project fell considerably short of its goals, despite a strong CS 350 team member and full support from the non-class portion of the team, due to technical problems beyond the scope of the student's project that were in no way due to the student. Furthermore the site of the work was the Computer Center, well away from the CS lab. Thus, the SIS student was present only during the meetings for the class's brief presentations, and there was a sense of disconnection with the rest of the class's efforts. We will think twice before allowing a remote project in future offerings of CS 350.

Conclusion

We taught a course CS 350, *Advanced Team Project*, in which one professor directed several diverse team undergraduate research projects with a heterogeneous class of students. This proved to be a feasible course to teach, at least when students have prior project and research-related experience, and our students succeeded with their project goals. Based on this experience, we have the following suggestions for future offerings of such courses.

1. A one-month block term offered advantages for this project-oriented course.
2. Regular meetings to hear brief presentations from teams may encourage interactions among teams, promoting a larger sense of a research community; but such meetings should not be too frequent and should focus on quality of presentations and peer questions.
3. Professors who participate in project implementation must beware of overcommitting themselves.
4. Electronically tracking students' time on task, together with team and individual reports, forms an effective basis for grades in such a course.
5. Projects where the whole team consists of class members appear to work better than projects where most of the work is done remotely.

A course such as CS 350 serves as a model for incorporating undergraduate research into the CS curriculum, and recognizing the directing professor's work as part of the teaching load.

References

Barnard, A., Eliason, M., Engle, R., Engle, T., Riewe, G., and Brown, R. A. (2004). Olafractal. Retrieved January 1, 2006, from `http://www.cs.stolaf.edu/projects/olafractal/`.

Braasch, D. (2006). Software transfer: A life and death issue for projects. In Bohy, J., editor, *Proceedings of the Midwest Instruction and Computing Symposium.*

Eliason, M., Engle, R., Engle, T., and Riewe, G. (2001a). Olafractal. Presentation in the St. Olaf College Mathematics Colloquium Series, Spring 2001.

Eliason, M., Engle, R., Engle, T., and Riewe, G. (2001b). Olafractal. Poster in St. Olaf Science Symposium 2001.

Etshokin, A., Goudzwaard, A., et al. (2006). Escher, the web portfolio manager. Retrieved February 1, 2006, from `http://www.cs.stolaf.edu/projects/Escher/`. Team project in courses CS 284 (Spring 2005), CS 390 (Fall 2005), and CS 350 (Interim 2006).

Etshokin, A., Handley, M., Middlecamp, D., Mueller, C., Stroh, J. V., and Hall-Holt, O. (2005). The palantir project. Retrieved January 1, 2006, from `http://www.cs.stolaf.edu/projects/Palantir/`.

Mueller, C., Brown, R., et al. (2004). Cpet, the co-process extension tool. Retrieved January 1, 2006, from `http://www.cs.stolaf.edu/projects/CPET/`.

Wiggly, G. et al. (2006). Northfield citizens online. Retrieved January 1, 2006, from `http://www.northfield.org/`. Website to form an online community for Northfield, MN.