Learning Environments for Requirements Engineering

Mildred L. G. Shaw and Brian R. Gaines
University of Calgary
Calgary, Alberta, Canada T2N 1N4
mildred@cpsc.ucalgary.ca, gaines@cpsc.ucalgary.ca
http://ksi.cpsc.ucalgary.ca/SERN/, http://ksi.cpsc.ucalgary.ca/KSI/

Abstract: The education of computer scientists addresses issues of system analysis, design and coding rather more than it does those of requirements. Since the essential issues of requirements engineering are those of managing a human community framing its needs and negotiating their satisfaction, it is difficult to develop a realistic learning environment in an educational context. This paper reports on experience in introducing requirements engineering projects in an undergraduate curriculum through involving students in role playing supported through a collaborative learning environment on the World Wide Web.

1 Introduction

The 'waterfall' model of system design has long been outmoded, if it ever has been more than a misleading ideal. The concepts that people can express their requirements for a system independently of the technologies that might satisfy those requirements, and can do so in a complete fashion that leads to a well-defined specification amenable to systematic analysis and design are both patently unrealistic. Requirements are most often conceptualized in terms of past or potential solutions, and requirements evolve as they are analyzed and as designs develop. Requirements engineering in the real world is essentially an evolutionary socio-technical process in which the expression of a need is matched by the suggestion of a technology that satisfies it, which then triggers the re-analysis, re-expression and reconstruing of the need, extending and changing it. The needs expressed by one user lead to the design proposals that trigger the expression of needs by other users, which makes a causal dependency model of a design rationale a misleading and incomplete model for the purposes of requirements tracking.

In an educational situation, the complexities of requirements engineering are either so simplified as to be meaningless or expressed in such complex socio-technical terms as to appear impossible to resolve. There is a danger that students come to see them as "soft" issues which are an unwelcome obstacle to the "hard" processes of system development, and never come to understand that the management of the combined requirements/design process is the key to successful system development, and one of the most rewarding and demanding aspects of the life of a computer professional.

This paper reports on a third year undergraduate course whose objective is to introduce students already skilled in analysis, design and coding to the issues of requirements engineering, in part through lectures, but in major part through projects which attempt to instantiate the issues of real-world software development.

2 A Collaborative Learning Environment for Requirements Engineering

CPSC 451 is a required third year software engineering course for all Computer Science majors at this University, is centered on projects that involve the students playing roles in teams representing customer and supplier organizations. There is a lecture component of 3 hours a week for 13 weeks, and a laboratory (practical) component of 3 hours a week for 13 weeks. The lectures are of standard format covering classical software engineering topics and methodologies, including the SEI levels of maturity and continuous improvement in an organization (Humphrey, 1989).

Each student is assigned to two different groups of 10 to 12 students. In one group she is one of the supplier team, and in the other one of the customer team for a different project. The process starts in the last 5 minutes of the very first class, when each customer group is allocated a project at random. The course web site gives a very short, informal and vague description of the problem. For example Project 1 in 1996 was:

Write a specification for a system to keep track of a business forms salesman. It must work out the routing (you may assume the route is linearly ordered). A list of which customers are likely to reorder must be produced, where they are located, what type of products they may want, and what reordering rates they have.

Each group gets a different problem, but they are all of a similar level of complexity. Each customer group has two days to prepare an informal requirements document for the project. They are subsequently responsible for its evaluation and criticism as it progresses; that is, they are the customer for the system. They are expected to be present at all presentations to ask questions, and comment on all the write-ups and documentation. Each grade, given by me not by the students, depends on how thoroughly the evaluation is carried out, the extent to which it is fair and reasonable and the extent with which it agrees with a well-founded methodology. Groups are advised to show all drafts to the teaching assistant (TA), and discuss any problems or disagreements. It is not very long after the start of the project that the customer and supplier groups reach the conclusion that the interchange cannot be done entirely by documents, and that they need to meet and negotiate problems, expectations, and what will be included in each version of the software.

The supplier group works with a different problem to receive informal requirements for a system, to produce a formal specification, a management plan, produce the analysis and design in the form of an overall and a detailed design document including test plans, a user manual, code a prototype, evaluate and refine the prototype, and present a final (prototype) product according to the details given. Public (to the whole class) oral presentations and discussions are required at various points within the project, and are evaluated and assessed by the customers. It is certainly not required, but usually the students will arrive in their best business clothes for the presentations, and fully enter into the roles they have been given. Every 3 to 5 days another part of the project becomes due for submission to the customers (and to me and the TA), and students are quickly made aware of the social pressures to conform to due dates. This may be the first time that any of them have considered that due dates are not altogether arbitrary, and that other people's deadlines depend on them. In turn, each student may be inconvenienced by other people's last minute rush to complete work, not only in the other group, but also in their own where, for example, an editor may require input from several people before the final document can be prepared.

No marks are given for coding. This is where much of the effort goes, and this practice seems to the students to be unfair. However, all the students have a thorough grounding in programming, and what they are learning through this course is the application of what they already know and is being covered in lectures, the management of their time, how their own working style fits into the range of working styles among their peers, and how their own time and work management affects others. Each student keeps a log of activities in the form of a diary with dates, what was done, and time spent on each item. At the end of each month, every student prepares a set of reports assessing each member of their supplier group. This requires a paragraph per person, including themselves, outlining who did what during the month, and how their work can be assessed, using some sort of grading scheme on one or more criteria. This means that they must get to know the people in their group as soon as possible, and decide how each has contributed to the group work during the month. This is not optional, but is a required part of the assessment to pass the course. In general the reports are thoughtful and responsible. The students do not always give themselves top marks, but say things like: "x did not contribute much to the group discussions — maybe I spoke too much of the time and didn't give x a chance"; or "I thought y was really stupid at first because his section of the documentation was such a mess — but I soon found out that he was a very good programmer and just had difficulty expressing himself in English".

In order to prepare students for the project, apart from technical considerations, one hour is spent talking about personality variables, the range of possible working styles, and how a person's strong points should be built on rather than concentrating on their weaknesses. A short form of the Myers-Briggs Type Indicator is administered for each student to find out their own personality preferences, and discuss how each type can benefit from input from other types. The students invariably get the message, without it being made explicit, that their job is to encourage the smooth functioning of the group and to involve every member as equally as possible without requiring everyone to take part in every job.

This is a course with a heavy workload, and students need a lot of encouragement, especially towards the end of term. The TA acts in an advisory capacity, making suggestions but not giving definitive answers, buffering them from any interference or undue influence on any of their decisions. The groups can use any methods, equipment, and language of their choice, but each must be justified as to its suitability for the project. There is no doubt, however, that after the course has finished the experience is highly valued, as an extract from a letter sent from an ex-student will demonstrate:

...although I didn't fully appreciate it at the time - much of what I learned in the class project...helped me to better understand and cope with the working world. For the last seven months I have been working as a contract programmer at Deutsche Bank's Regional Head Office in Singapore....I currently have to code according to design documents....work in a team of over 20 people all making changes to the system simultaneously, rushing to meet deadlines - like what I went through during the CPSC 451 project. Thanks for all you've taught me....

It is not necessary to make explicit psychological and sociological perspectives part of the academic curriculum in CPSC 451. In any event, the science curriculum to which most of the students have been exposed encourages linear thinking and objectivist values, and is a poor foundation from which to understand industrial issues and practice. The students *experience* the significance of roles, construct systems and inter-personal interactions. The alternation of their own customer and supplier roles brings them to terms with the nature of construct systems, both their subjective artificiality and their ethical implications in terms of role consistencies, responsibilities and accountabilities. Being responsible for conceiving and articulating *requirements*, in particular, is a new experience for most students, and leads them to be more thoughtful about how those requirements arise.

3 Requirements Tracking Through the Web

One of the problems of running the projects effectively in the early years of CPSC 451 was, while the students moved through the stages of requirements evolution and learnt lessons at each stage, there was no 'corporate memory', 'requirements tracking' or 'design rationale capture' except in their own memories. It was difficult for students to reflect on the entire experience and learn the larger lessons that come from a global perspective of the interactions within and between the customer and supplier teams.

It has been natural to introduce the use of the World Wide Web into the operation of CPSC 451 to support the group collaboration and the customer-supplier negotiations which in previous years were communicated only through documents. All the course materials are already made available to the students through the web, and it is a simple extension for students to place the documents relating to customer requirements, supplier comments and proposed solutions, the ensuing negotiations, and descriptions and screen dumps of the developed applications on the web.

The following sub-sections reproduce the notes given to students on how to use World Wide Web facilities in a simple and natural way to manage their projects.

Using the Web for Requirements Tracking

The World Wide Web supports the sharing of structured multimedia documents with links to one another and to other material, not necessarily in HTML format. This makes it a suitable base technology on which to develop a requirements tracking system for software engineering that tracks the relationship between requirements, analysis, design, implementation, application and maintenance.

There are complex requirements tracking tools being developed that use the full client-server capabilities of the web to provide highly integrated systems. For CPSC 451 we will use the web more simply to develop a set of linked documents tracking the customer group requirements specification, the negotiation with the supplier group to clarify them, the design specification, the customer commentary on this, the implementation, acceptance procedures, and so on.

Rather than use special-purpose servers we will take advantage of the web's capability to link documents held in different locations, and generate a sequence of documents held in the directories of those in the customer and supplier groups. A single master index managed by someone in one of these groups will be used to make the distributed set of documents available to everyone in the groups.

Linking and Annotating Material

By putting all the customer and supplier documentation in the web directories of members of the customer and supplier teams it automatically becomes available to everyone. However, it is not linked together and it may not be obvious how the documents relate to one another. Two things can be done to improve the situation.

First, someone in the customer group should agree to become documentation manager for the entire project and maintain an index in their web directory of all the documentation from both customers and suppliers. This index will contain pointers to the distributed set of documents allowing them all to be accessed from one place.

Anyone needing to look at the project documents will start with the index and click on a document in it to fetch that document. Anyone generating a new document that should be entered in the index should email its url to the documentation manager specifying where it belong in the index.

Second, while you cannot edit a document in someone else's account, you can fetch it with a web browser, save its source, annotate that source with your comments, and make the annotated document available in your account. Thus, the index will grow to be a sequence or a tree of documents such as a customer specification, a version annotated by the suppliers with queries about the specification, a version of this annotated by the customers with answers to queries, and so on. Similarly, an initial design specification from the suppliers can be annotated by customers.

Style of Annotation in HTML

You will need to establish some style guidelines for documents so that you can distinguish annotations from the original. One good technique is to use a horizontal rule before and after annotation. Tags for bold and italic can be used to mark particular sections of text to which you are making reference. There is no obvious or standard way of doing this. Experiment with HTML and decide on an approach for your group (but don't spend more time on an annotation scheme than you do on the actual software engineering project!).

What Should be Documented on the Web

Any documents that you can generate on paper you can also put up on the web, so that your project web should finally provide a complete account of all the activities of the project. For example, you can link to design material that includes screen-dumps as GIF images. In a real-world project that used the web itself to implement a client-server system you would even be able to link to the working system so that the requirements and design information was available for maintenance and enhancement support. If your implementation is on the departmental unix system then you should be able to provide a link that starts a live demonstration.

4 Case History—Project 1 Business Forms Salesmen Tracking

The task definition for Project 1 was given at the beginning of Section 2. Figure 1 shows the supplier group's home page—both customers and suppliers generally invent company names, structures and logos to provide a realistic infrastructure for their activities. The lower part of this page (not shown) gives the project structure as: secretary and contact; 2 editors; document manager; a coordinator and 6 members as user interface design team; and a coordinator and 3 members as database design team.

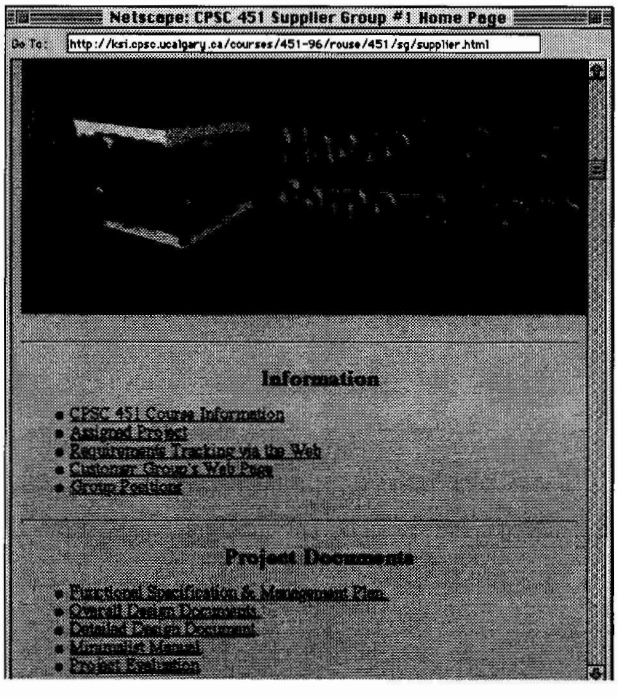


Figure 1 Home page of supplier group

Figure 2 is a concept map of the major part of the web structure developed for the project created automatically through our KMap tool (Gaines and Shaw, 1995). The nodes are the titles of web documents, and the arcs show hypertext links between them. The customer and supplier groups have each established home pages with links to their own documents and to relevant ones in the other group.

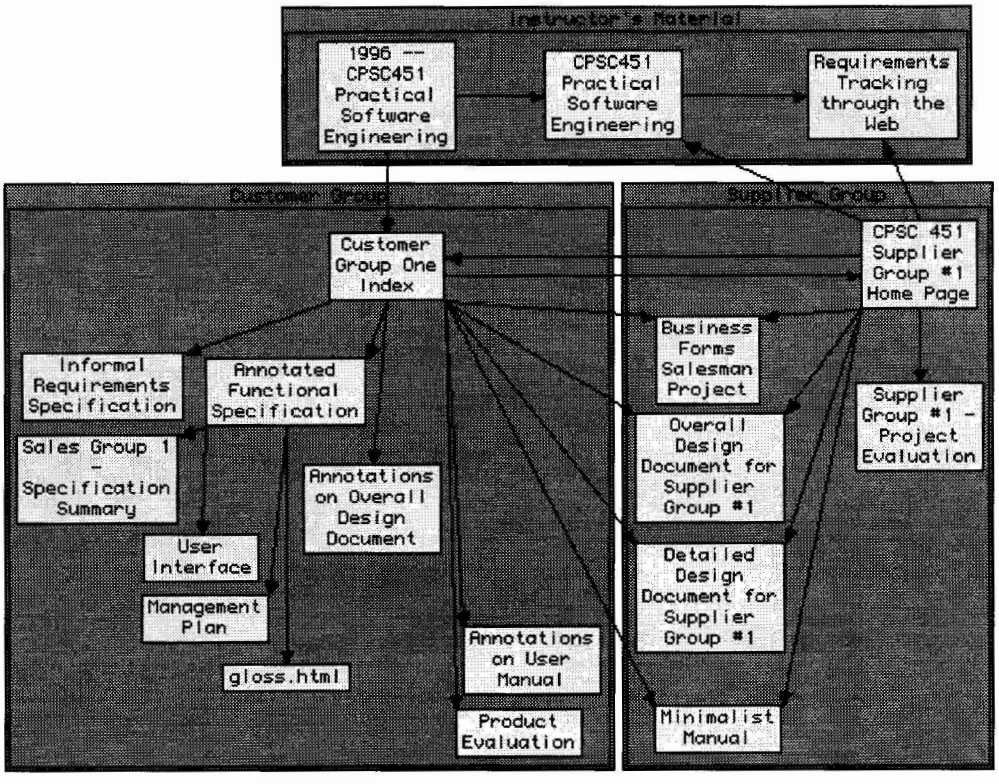


Figure 2 Concept map of documents and hypertext links in customer and supplier webs

The customer group developed an informal requirements specification to which the supplier group responded with a functional specification for a "Business Forms Salesmen Project." The customers took this and annotated it (Figure 3). The suppliers used the annotated document to generate an overall design which the customers again annotated, leading to a detailed design and user manual, again annotated, and to a customer product evaluation and a supplier project evaluation of lessons learnt.

Figure 3 shows the customer annotation of part of the proposed user interface in the supplier's functional specification. The upper part is copied from the supplier's document and the lower part is added in a different type face by the customers.

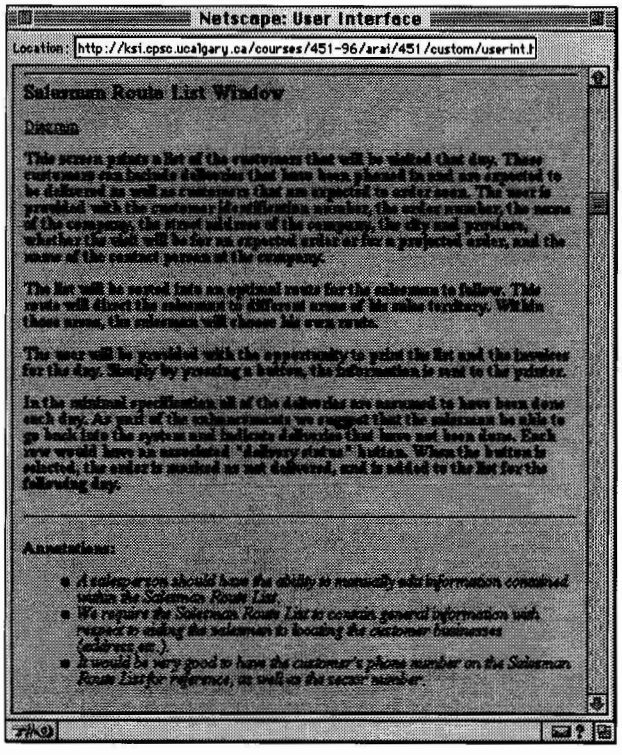


Figure 3 Customer annotation of supplier functional specification

Figure 4 shows the diagram produced of the proposed user interface when the link to "Diagram" at the top left of Figure 3 is followed.

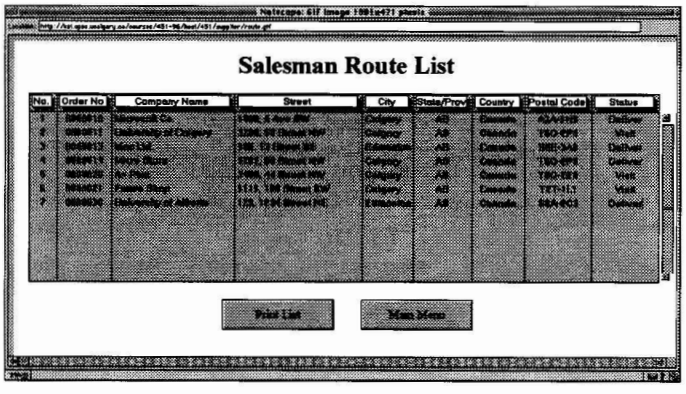


Figure 4 Supplier diagram of screen layout

Thus the two groups were able to communicate their designs and coordinate their negotiations using standard web tools in a simple and natural fashion.

5 Student Evaluation of Lessons Learnt from the Project

Part of the task of the supplier group is to write an evaluation of lessons learnt and place it on the web:

> *The final document required is a project evaluation (about 10-15 pages) which should include a discussion of how well your project satisfies your original functional specifications or what has been added or subtracted; how useful the design process was and what differences (if any) you propose for the design assignment; how the test plan worked, how the real management structure resembled the planned one; comparison of actual times taken for each part; how you would have done it differently if it were a "real" project (dissenting opinions in the group may submit an additional statement); and in*

what way you could make the project less work if you were to do it again. Include a section on "lessons learned" from doing this project.

The following subsections are extracts from this evaluation for Project 1.

The Good (What went right):

- Fortunately, we all seemed to get along with one another, and we seemed to avoid any personality conflicts which could have had a detrimental effect on our progress. The members of the group usually communicated well with one another.
- The members of the group had a high level of commitment to the project, so everyone did the work assigned to them. With the exception of losing a group member and not finding out until weeks later, people finished what they said they were going to do. It was good to know that we could depend on people, since one person messing up would have reflected poorly on the entire group. Despite the fact that each of us had a different schedule and other commitments outside of this project, meetings were well attended, and we usually had little trouble finding time to work on the project.
- Everyone stayed involved throughout the project. Since almost everyone was involved to some extent in each phase of the development, we didn't have to spend a lot of time familiarizing people with recent developments.
- The loss of a group member didn't slow us down. The remaining members managed to pick up the slack and make up for the missing person.

The Bad (What went wrong):

- The group lacked clear leadership and structure. We had a wide range of different skills and talents in the group, but unfortunately we had no one who took charge immediately. There was no one in the group who seemed comfortable assigning work to others, especially since all of the group members were essentially equals. It may have been a good idea to choose a leader, although it seems unfair to thrust leadership upon a reluctant victim, and a reluctant leader may have been worse than no leader at all.
- Sometimes people were reluctant to volunteer for tasks. Again, having a leader to assign tasks would have been good. This may have ensured that the workload was spread more evenly amongst the group members.
- Lots of redundant or unnecessary documentation. At times, we felt that meeting the length requirements for certain documents meant that we had to pad our documents with information that wasn't really necessary or appropriate. The time spent adding to the documentation might have been better spent doing other things.

Lessons Learned

- Clearly define a group structure early in the project. This would probably include choosing a group leader, and possibly breaking down the group into smaller subgroups. It is very inefficient for the entire group to participate in each aspect of the project, and meetings of the entire group are often long and tedious. Smaller groups are better able to find time to meet.
- Make sure that as many people as possible are familiar with the chosen programming language. It should be possible to choose a language fairly early, so if only a few people know the language the rest should attempt to gain some experience with it.
- Define standards for formatting documentation. Since it is impossible for one person to do all of the documentation, several people will be contributing to each document. If each person uses their own set of rules then the editor will have to do a lot of extra work to combine the sections into a consistent final form. It creates a lot of wasted time if several people have to write and rewrite the same thing.
- Set a structure for meetings. Regular meetings may be necessary. Free form meetings are a bad idea. There should be an agenda so that everything that needs to be discussed gets taken care of, and everyone knows what they need to do to prepare for the meeting. After the meeting everyone should be aware of their responsibilities.
- Set deadlines and stick to them. Since many parts of the project pass through several sets of hands on their way to completion, it is essential that each step along the way is reached on time, or else the entire project could be delayed. Make sure to allow flexibility in case of emergency, however.
- This project proved to be a good experience for us. Most of us didn't have much experience working in a group of this size, so there were lots of unanticipated problems that arose. As the project progressed we became more adept at dealing with these problems.

6 Conclusions

Experience with CPSC 451 indicates that it is possible to simulate a situation in undergraduate education that provides a meaningful and effective learning environment in which students can come to understand the socio-economic dynamics of requirements engineering through experiential, collaborative learning. Playing the role of a member of a customer or supplier team is one that comes naturally to computer science students, and the simulated situation rapidly becomes sufficiently 'real' for the experience to be meaningful and effective in supporting learning.

The World Wide Web provides a supporting technology that is very simple and already widely available to educational institutions. CPSC 451 used only standard web servers, browsers and text editors. No special-purpose tools were used. The students had familiarity with computer use and text editors, but most had no prior experience in the use of the World Wide Web. We have found in other courses with students across a range of non-computing disciplines that the capability to prepare documents for the web is easily acquired in a few hours experience. The approach described in this paper can be used in any discipline.

The web created by CPSC 451 students in the winter semester of 1996 is part of the learning environment supplied to CPSC 451 students in the winter semester of 1997. Thus collaborative learning is being encouraged not only within cooperative groups, and across negotiating groups, but also longitudinally between classes in different years. The students who are learners in one year know that they are also teachers for the next, and hence can take pride in knowing that their achievements will be of value to others, not just exercises for the instructor to assess and then forget.

CPSC 451 is an important course for students seeking employment in industry, since most potential employers see computer science students as lacking in the group collaborative skills that CPSC 451 develops. The archiving of the materials on the web automatically creates a "portfolio" for each student through which they may show to a potential employer that they have had experience both in specifying requirements for others and in developing systems through teams that have negotiated requirements with others.

Acknowledgments

Financial assistance for this work has been made available by the Natural Sciences and Engineering Research Council of Canada.

URL's for Materials

CPSC 451 archives for 1996: http://ksi.cpsc.ucalgary.ca/courses/451-96/
Related articles: http://ksi.cpsc.ucalgary.ca/articles/

References

Gaines, B.R. and Shaw, M.L.G. (1995). WebMap: concept mapping on the web. **World Wide Web Journal** **1**(1) 171-183.

Humphrey, W.S. (1989). **Managing the Software Process**. Reading, Massachusetts, Addison-Wesley.