Live-Through Case Histories as Motivation for Software Process[1]

Lawrence Bernstein and David Klappholz
Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ
(lbernstein@ieee.org, d.klappholz@worldnet.att.net)

1. The Problem

"Software Crisis" is a cry often heard in both the popular and the trade press. It is well known within the software community that far too many software projects fail. Some fail to meet stakeholders' functional requirements, often because not all stakeholders have been identified and involved in the requirements analysis process [Royce1975]. Some fail to meet load requirements because mathematical analysis to predict performance has not been done. Insufficient testing, often the result of unrealistically tight schedules, causes far too many projects to be released with faults which turn into failures at inopportune times. Finally, far too many software projects are cancelled before completion because of inadequate -- or no -- cost, personnel, and schedule estimation. Software project failures all too often result from lack of training in, or lack of acceptance of, Software Process (Best Practice) on the part of their managers [Lederer1995]. A likely source of the problem is, therefore, to be found in the training of those who become software project managers [Paulk]. Since software project managers are recruited from the ranks of experienced software developers, and since one typically prepares for a career in software development by going through an academic program in Computer Science, a likely source for the problem's cause, and, hopefully, its solution, is the university.

Most computer science faculty are concerned with advancing the State of the Art, i.e., software technology. Many, having, themselves, little or no experience in medium- or large-scale team software projects, are effectively unaware of, and, at best, only mildly interested in the State of the Practice, i.e., the software development process. Many, unfortunately, deem courses in Software Engineering to be of intellectually low caliber, and of little value. Many are, unfortunately, correct in one sense, i.e., that Software Engineering courses, as well meaning as are those who teach them, often fail to reach students with their critical messages. But why is that the case?

The typical Computer Science student is trained in the solution of small well-defined problems whose solutions provide immediate feedback. A code module to solve a well-defined small-scale problem of the sort usually given as homework in the first two or three years of a typical undergraduate program in Computer Science is easily written by a reasonably competent student; it is easily tested via compilation and execution with sample data. The typical Computer Science student has considerably above-average skill at this type of endeavor; s/he has, in fact, probably decided to major in Computer Science for that very reason. In most cases, though, s/he has less than average skill in handling ill-defined problems, problems whose solutions provide only long-term payoffs, and, of course, problems in group dynamics.

The typical Computer Science student baulks at the prospect of being forced to follow what appears to be a rigid, constraining, process, and, worse yet, one which is far less fun than "hacking" code. In fact, many Computer Science students, even those entering a required year-long Senior Project course, like the one in which the authors are involved, will deny, often passive aggressively, but frequently verbally, the existence of anything but technology expertise as a requirement for success in team software development; students often defensively view a course designed to teach the principles of software process as irrelevant or trivial, and of no use to those, like themselves, with highly specialized technological expertise. In the worst case, students resent being required to take such a *soft* course; they're certain that they can successfully complete a yearlong five-to-ten-person team project as successfully as they've completed countless personal projects without the need to learn anything, except, perhaps, a bit more technology.

---

[1] This work was supported by a grant from the New Jersey Commission on Science and Technology.

2. The Proposed Solution

The challenge to those who teach Software Process is to overcome the natural biases of their students. We chose to *shock* ours out of their complacency and ignorance. As is suggested in the title of this paper, we did so through the use of software project case histories. The case-history approach is time-honored in such process-oriented and people-oriented disciples as law and business. But our approach is not one of simply recounting case histories in class or of requiring students to read case histories at home. The reading of case histories of failed software projects tends to convince Computer Science students of others' stupidity; it may even convince some, intellectually, of the existence of the problems of which we are attempting to make them aware. Reading, however, is unlikely to convert many at the gut level, the level at which one sits up, takes notice, and changes one's ways. Our approach is, rather, to force students to live through specific case histories, each one chosen to get across a small number of important issues. It is based on the notion that if one is forced to exhibit one's own mistakes in public, in a non-threatening situation, one realizes immediately that there is something to learn, and is likely to be motivated to learn it.

The live-through method works as follows. First, a set of software process issues to be motivated is selected. Some sample issues are:

-the need to perform rigorous requirements engineering [Vinter2000]

-the need for up-to-date documentation of a project in progress

-the need to identify risks [VanVliet2000] and to develop contingency plans

-the need to evaluate a design's complexity, using function points or some other method, in order plan personnel requirements ...

Second, a case history is chosen in which the project failed, either entirely or in part, because one or more of the chosen issues was ignored or executed badly. Students are not given the entire case history up front; rather, they are given the same problem/project as was given the developers who executed the case history. They are given no more information about the problem/project than were those developers.

In the required two-semester Senior Project course in which the authors are involved, students are required to spend the bulk of the year on team projects. If these projects are to be done with an awareness of Software Process (Best Practice), students must be shocked into an awareness of its existence and necessity in a short time. In the balance of this paper we discuss our first experience with the "live-through" method, our thoughts on how we will use the method as a primary tool in a new Software Best Practice (process and tools) course which will, eventually, precede the Senior Project course, and on the university consortium which we have begun to build for the purpose of finding and fine-tuning live-through case histories and for performing large-scale studies of the efficacy of the method.

3. A First Live-Through Case History: Administration and Initial Evaluation

Because there would be only one live-through case history in our Senior Project course, we had to choose one that would achieve the greatest effect in the limited time available. We chose the case history of a brief development project that one of the authors worked on, in 1985, as a public service project. The major issues involved in the chosen case history were:

-the need for a serious requirements engineering process

-the need for clearly written documentation of all work done at every stage of a development project [Paulk]

-the need to plan for unexpected contingencies [McConnell1998]

Some additional issues were:

-the need to explore design alternatives

-the need to account for human foibles

The problem/project, which we presented to students at the start of the two-semester Senior Project course is shown in Figure 1.

Problem: An elementary school of 500 students is using a manual method for tracking books on loan. As books are taken from the library the book card is taken from the book jacket and filed by date. Books may be borrowed for two weeks. As books are returned the card is put back into the book jacket.

Books that are not returned in two weeks are considered overdue and an Overdue Book Notice is sent to the students. Students who do not return a book within three weeks are given a second notice. Books not returned in four weeks have a third notice and are reported on a special 'Critical Overdue' librarian's report and the book card is given to the librarian. Clerks write overdue book notices on one half of a sheet of paper for about 200 books each week. The notices are distributed weekly. This layout of the Overdue Book notice was used for many years:

Glenwood School Overdue Book Notice

Book Title:

Student Name:

Teacher Name:

Date of Notice:

Notice 1 2 3 (circle one)

This book is overdue please return it promptly

Project: Automate the overdue book notice process using a computer borrowed from the computer class. This is the first computer automation project in the school. Data may not be left on the computer from week to week. The computer is not networked.

Figure 1

The requirements engineering lesson was learned the hard way by the original case history's development team. The original Glenwood School Overdue Book Notice Project had a hidden requirement, one that was so obvious to the customer, the Glenwood School librarian, that she failed to mention it to the team. What the librarian forgot to mention is that overdue notices must be sorted, first by teacher name, then, for each teacher by class, and, finally, within each class by student's family name. As is often the case in the "real world," the work product was rejected by the customer because the original developers failed to elicit the hidden make-or-break requirement, and, therefore, failed to satisfy it.

If notices are not sorted by teacher name, then the librarian, who hands notices to teachers for distribution to students, must sort them by hand, thereby vitiating a good part of the benefit of using an automated system; under these circumstances, the librarian will simply not use the automated system. If each teacher's notices are not sorted by class, then the teacher must sort them manually, thereby vitiating a good part of the benefit to the

teacher; under these circumstances, the teacher will not use notices generated by the automated system. Teachers, like many other workers, are naturally resistant to changes in long-standing routines; as a result, if the format of the overdue notice generated by the automated system is at all different from that of the handwritten notices produced by the librarian, or if the notices for each class are not sorted by student name, as are the handwritten notices, then teachers will resist using notices generated by the automated system. In the actual case history the Glenwood School principal was ambivalent about the adoption of this new system and did not make its use a requirement for the teachers.

The need for high-quality documentation and for contingency planning, the second and third intended lessons of the Overdue Book Notice case history, were motivated for students by the classroom equivalent of the real-world phenomenon of loss of staff members due to illness, death, relocation, etc; at the midpoint of the project, the one student from each team judged, by the instructor, to be the team's strongest developer and another, randomly chosen, team member were removed from the team and re-assigned to a different team.

During the course of the live-through case history, the instructor played the role of the customer, the Glenwood School Librarian. The class of approximately forty students was divided into four development teams of approximately ten students each. Students were told that their work would be evaluated, by the customer, exactly as work is evaluated in the real world.

Having devised the initial live-through case history, how did we propose to evaluate its success? Our initial impulse, when we conceived the idea of live-though case histories, was to subject students to entry and exit questionnaires, with results to be measured by the degree of meaningful change between answers to process-related questions in the entry questionnaires and answers to the same or similar questions in the exit questionnaires.

CS460-461 Entry Questionnaire

1. List each CS and Math course which you have completed (course number and name, please) and your grade in the course.

2. List each Humanities course you have completed (course number and name, please) and your grade in the course.

3. In which high-level programming language(s) (C, C++, Java, Pascal, Fortran, Cobol, other) are you comfortable?

4. In which Scripting language(s) (Perl, Javascript, other), if any, are you comfortable?

5. In which other language(s) (HTML, XML, SQL, other), if any, are you comfortable?

6. In which additional software technologies (CORBA, COM, DCOM, other), if any, are you comfortable?

7. List each significant one-person software project you have done -- in a course or elsewhere.

8. For each significant one-person software project which you have done:

-Give a clear description in English of what the software was intended to do

-What language(s)/technology (ies) were use to implement the software?

Figure 2

In addition to process-related questions, the entry questionnaire was originally to have contained questions relating to previous experience, with the idea that there might be a significant correlation between a students' previous experience and both the student's responses to process-related questions and the student's sophistication in performing the live-through case history. Aspects of previous experience would include software projects done in the context of courses taken, co-op assignments -- approximately one third of our undergraduate computer science students are in a co-op program -- summer jobs, etc. Course-related information would include lists of courses taken in computer science, in mathematics, and in the humanities, along with grades in those courses. The thought was to correlate measurable academic achievements in "practical" computer science (introductory programming, introduction to data structures, etc.), in formal reasoning (discrete mathematics, analysis of algorithms, theory of computation, etc.), and in communication skills (humanities), with performance in the live-through case history.

Because one of the authors felt that process-related questions on the entry questionnaire would bias the results of the live-through case study, we decided not to include such questions, but, rather, only experience-related questions. Evaluation of results would have to be done differently than was initially planned. The entry questionnaire, as distributed to students before the start of the live-through case history, is shown in Figure 2.

The first meeting of the Senior Project course was held on Tuesday, August 29, 2000. Entry questionnaires were e-mailed the same day and students were asked to fill them out and to return them by September 14, 2000. The Overdue Book Notice problem/project was handed out and team assignments were made. The first team meetings were held on Thursday, August 31, 2000. The course met Tuesdays and Thursdays, with Tuesdays devoted to lectures on various software engineering-related topics and Thursdays devoted to project workshops. Each student was required to maintain a spiral-bound notebook for class notes, project meeting notes, notes from assigned readings, personal time records, resources used, and homework solutions. Notebooks were checked randomly during Thursday workshops.

Topics of Tuesday lectures included:

1. Mars Explorer Case History [March1999]

2. Gantt Charts [VanVliet2000]

3. Software Architecture

4. Software Process Models [Paulk]

The live-through case history lasted for a total of six weeks, with students asking questions of the customer during class and by e-mail; all questions and all answers were shared with all members of all teams. At the end of the first three weeks, on Thursday, September 21, 2000, staff changes were made [McConnell1998] [DeMarco1995]. The two people selected by the instructor from each team, as indicated above, were rotated to a different team.

Our first indication that the decision not to rely on take-home questionnaires to evaluate the impact of the live-through case history was a good one came from the fact that only two thirds of the students bothered to return and complete them. All students were aware that the questionnaires would count toward their grade, but many presumably viewed them as just another homework set and figured that missing one wouldn't lower their grades sufficiently to be of concern. Far worse, though, was the fact that many of those who filled out and submitted the questionnaire put little effort into their responses, but, rather, responded with what they guessed the instructor wanted to hear.

Evaluation of the efficacy of the live-through case history began, shortly after the staff changes, with a single question on the first hour exam, the question shown in Figure 3.

If you knew about staffing changes of Thursday September 21 before they happened, give three things that you would have done differently in your work on the library overdue notice project?

Figure 3

The wording of the question suggested to some students that the intent of the question was

-either: Had you been informed about the imminent staff changes a short time before they were put into effect, what would you have done to prepare for them?

- or: Had you been told at the beginning of the semester that staff changes would be made on September 21, what would you have done differently from the very start of the project?

What we had in mind was actually:

Had you really believed, from the start of the project, as you were told, that it would be run as a "real world" project, and had you realized that in the real world people often leave projects with little or no advance warning, what would you have done differently from the start?

Even so, the majority of students offered responses that suggested that they had learned a significant lesson from the first half of the live-through case history. The most interesting and pertinent lessons may be divided into four categories: change in documentation, change in staffing policy, change in overall approach to the problem, and change in planning for potential staff changes.

twenty-three of the thirty-four students who took the exam recognized the critical importance of documentation and realized, by having suffered through the consequences, that they or their teams had done an inadequate job. They indicated that they would have personally documented their own work better than they had, that they would have insisted that the entire team document its work better than it had, or both.

Sixteen responses[2] indicated that project staffing would have been different from the very start of the project. Eleven responses were to the effect that more staff members would have been assigned to some or all of the sub-tasks. (One student used the word "understudy.") Five responses suggested either that work would have been "evened out" more than it was, or that one or two students would not be allowed to do the bulk of the work. A few students clearly learned the danger of letting one or two students take over the bulk of a team's work, either through assertion of actual superior ability (guru-hood) or through the use of unwarranted bravado. One additional response was that the people transferred had bullied the team into accepting their approach to the problem. The bullies had been divisive and had done the team a favor by leaving.

Seven responses indicated an overall change in approach to the problem. Three suggested that better planning would have helped, though "better planning" is vague enough that the respondents may have actually had nothing specific in mind. Four suggested, extremely perceptively in the opinion of the authors, that the decision on technology (presumably COTS) and choice of major implementation language would have been delayed as late as possible, either to simplify the implementation or to accommodate staff changes. At least two of the four clearly articulated the notion that the architecture/design should have been more "plain vanilla," i.e., that it should have been more conceptual than technology- or language-specific.

Finally, seven responses suggested that more time and thought would have been devoted to planning for the involvement of potential latecomer staff members. Three students suggested thoughtful concern and planning for their old teams should they have been the staff removed from those teams, and only one student -- one who was actually moved -- showed utter disregard for the old team and concern only for his own future prospects.

[2] Note that each student was asked for three responses; not all responses are tallied here as a significant number of students responded with one or more points judged as irrelevancies by the authors.

Only one student suggested that his team had done everything just right, one suggested nothing relevant to the question at hand, and three students produced responses that were either attempts at humor or the results of having misunderstood the question. (One indicated that the team would have hidden the identity of its guru; one suggested that the staff members who would be leaving would have been entirely ignored by the rest of the team from the start; the third suggested that the staff to be moved would have been asked to sign a non-disclosure agreement.)

It is fair to say that the first half of the live-through case history had a reasonable, if not profound, impact on at least two thirds (67% = 23/34) of the students. A few students, perhaps ten to twenty percent showed exceptional insight. But this simple question on the first hour exam was only our first attempt to assess the efficacy of the live-through methodology, and reflects results only up to the mid point of the case history.

The Overdue Book Notice live-through case history ended on October 17, 2000 when one team presented its product to the class and described the process they had used. On October 19, 24 and 26 the other three teams presented their products, discussed the methods and processes they had used in developing them, and answered questions from the instructor and from the rest of the class. All four teams had failed to identify the hidden requirement and the customer rejected their projects. (Grades were, of course, meted out on a more liberal basis, a basis having to do with quality of the product and degree to which students had learned Software Best Practice -- requirements engineering excepted -- in its development.)

Shortly after the debriefings, each team was required to submit, by e-mail, a written assessment of its work product and of the methods and processes it used in producing that product, as well as assessments of the products, processes, and methods of the three other teams. All four teams' assessments were circulated to all members of the class. It will be recalled that our first attempt to allow students to reply to questions at leisure -- in the form of entry questionnaires -- produced disappointing results. As indicated above, "it was clear that many of those who responded tried to anticipate answers the instructor wanted."

4. A First Live-Through Case History: Further Evaluation

Upon receipt of the written assessments, we realized that evaluation of the success of the experiment of using a live-through case history would be grossly unsatisfactory if it were based upon reports written at leisure by individual students or by groups of students. Three of the four reports showed little or no attempt at incisive analysis, but, rather, consisted largely of catch phrases that the students apparently thought the instructor wanted to hear. One report was encouraging as it dealt reasonably aptly with software process; it may be found in the Appendix to this paper. The rest of our evaluation was, therefore, based upon a sequence of questions on the second hour exam. The questions on the second exam were based upon yet another case history, this time somewhat reduced in scale from one which one of the authors managed in 1987. It is the case history of a telephone company service request system, and is shown in Figure 4.

BU&U is a statewide company with headquarters in Sunbathers Heights, NJ. Part of BU&U's business is satisfying service requests (for equipment installations, repairs, etc.) at customer sites. Each such request is handled by dispatching a BU&U technician to the customer site, from the BU&U Work Center nearest the customer location. BU&U has ten Work Centers. Until recently, the Service Request System was completely manual. As a potentially cost-saving measure, the VP for Customer Service at BU&U contacted Lawrence Amberstone, boy wonder Director of BU&U's Software Development Team (SDT) to see if the SDT could design and implement a computerized version of the Service Request System.

Figure 4 Part I

The Service Request System operates as follows:

1. Customers request on-site service by phone. A transaction representing a service request is entered into BU&U's Central Service-Request Computer, in Sunbathers Heights as it is received, over the phone, from the customer. Each transaction requires some processing by the central computer.
2. Customers sometimes call to inquire about the progress made on their as-yet-unsatisfied service requests. Each call requires that an average of 2 inquiry transactions be input to the Service-Request Computer.

3. Each Work Center has 10 clerks. They process, on average, 4 service requests and 2 inquires per hour. Each Work Center is open daily from 8:00 AM until 10:00 PM.

4. At each BU&U Work Center a designated employee, called the Dispatcher, is responsible for scheduling service visits to customer sites by BU&U technicians. Each Dispatcher is given a new PC, connected via BU&U's intranet, to the BU&U central Service-Request Computer. The way the Dispatcher finds out about service requests for which his/her Work Center is responsible is by periodically requesting Service Request Reports from the central Service-Request Computer. A Service Request Report is a list of service requests that must be satisfied within 48 hours by the Work Center that requested the Service Request Report. Each work center is expected to request 2 Service Request Reports a day.

5. The Dispatcher uses the Service Request Report to dispatch specific technicians to specific customer sites, to decide which, if any, technicians will have to work overtime so that all service requests can be satisfied on time, etc.

6. At the end of each day a Daily Transaction Profile (DTP) is sent back to the SDT for analysis. A DTP is a list of all the transactions received and processed by the Service-Request System that day. For each transaction, the DTP includes, the transaction's type (service request, inquiry, etc.), its arrival time, and the time at which it's processing was completed.

The SDT has developed requirements for the Service Request System, and has, through the use of prototyping, developed a design. Both the requirements and design documents have proved satisfactory to the VP for Customer Service, and she signed off, officially accepting, them.

Finally, Amberstone's SDT has implemented and tested the Service Request System. The requirements document has been consulted to determine the server and PC configurations necessary to easily handle the projected transaction traffic as well as to produce the average of two Service Request Reports that will come from each Work Center each day.

The Service Request System (SRS) is scheduled to be tested, off-line, at the Work Centers for one week, to be brought online with one Work Center the second week, with a second Work Center the third week ... and with all ten Work Centers on the eleventh week. Until the end of eleventh week the manual system will be used for those Work Centers not yet online.

Recap of the customer site plan:

week 1 acceptance testing

week 2 beta testing with the first Work Center

week 3 operational use with an accepted system supporting Work Centers 1 and 2.

... and so on.

Figure 4 Continued (Part II)

The system is delivered, and, at the end of the first week, acceptance testing went smoothly. No bugs were reported and system performance was well within specified requirements.
The first center is put online at the start of the second week. The system performs well.

The second center is put online at the beginning of the third week. At the end of the third week, everything has gone smoothly, except that on Wednesday of that week the Operations Manager, the BU&U employee responsible for all ten Work Centers, reports to the SDT that the inquiry transactions are taking longer to be processed than they were during the first and second weeks. As far as the SDT can tell, all transactions are being processed well within the response time specified in the requirements document, but the average transaction is taking somewhat longer to be processed than it did during the previous weeks. There is no explanation for this subtle increase in response time. Other work demands distract the SDT and they do not pursue this glitch. So the Operations Manager signs the software acceptance form and praises Amberstone for a job well done -- on schedule and within budget.

At this point the SDT is diverted to another project and does no further work on the Service Request System.

The third Work Center goes online on the first day of the fourth week. On the second day of the fourth week, Amberstone receives an irate phone call from the Operations Manager. "The system died yesterday at about 3:00PM, and nobody at the Work Centers has any idea why," says the Operations Manager. "When the system was restarted, it worked, but very sluggishly; transaction response times were sometimes not within specification. The system returned to satisfactory transaction times late in the day." The Operations Manager hints to Amberstone exactly how she would like to re-arrange his anatomy, and demands an immediate solution or she will return to the old manual system. "We can't run our business this way," she says.

You are SDT Director Amberstone and you reply, "This is a surprise to me. We noted some slow response times two weeks ago, from that week's DTPs, but have not been able to explain why. I will look into the situation and get back to you in two hours."

Figure 4 Concluded (Part III)

At the start of the second exam, students were given the case history of Figure 4 together with the single question shown in Figure 5.

1. What are the first things you, as Amberstone, do to try to solve the problem?

Figure 5

Students answered question 1 and handed in their answers before being given a summary of what Amberstone actually did, along with another question. (See Figure 6.)

1. What are the first things you, as Amberstone, do to try to solve the problem?

What Amberstone actually does is to drop all other activities and to consult the SDT and the support staff; he finds that they did not even know that the problem had occurred.

Figure 6 Part I

He declares the problem to be Priority 1, and focuses his entire staff on it. The staff runs the previous day's DTP and they find that the DTP ends just before the system hung. What there is of the DTP shows that the system performed normally until the point at which the DTP ended, so Amberstone's staff still has no idea what happened. Because there is no reason to believe that the crash resulted from a software error, Amberstone does not start a frantic debugging effort.

Rather, he dispatches staff to the four online Work Centers to gather more data. He then calls the Operations Manager, reports on the actions taken, and arranges to visit one of the sites of the Service-Request System the next day.

2. What do you, as Amberstone, do when you visit the site?

Figure 6 Concluded (Part II)

After answering question 2 and handing in their answers, the students were given another copy of the material contained in Figure 6, plus the additional material, including three additional questions, shown in Figure 7.

2. What do you, as Amberstone, do when you visit the site?

Amberstone looks over the data collected, reviews status with his people on-site and calls the other two sites to make sure his people are properly deployed. He also checks with his people observing operations at the computer center. He then observes the use of the system by listening to customer phone calls and seeing how the transactions are entered into BU&U's Central Service-Request Computer. He gets frequent updates on computer CPU utilization and response times. At about 3PM on the day of your visit the system hangs again.

3a. What is the most logical question for you, as Amberstone, to ask the people at BU&U headquarters and at the Work Centers?

3b. What answer does Amberstone get, from BU&U staff, to the "most logical question" that he asked.

3c. What short term actions do you take?

Figure 7

As the next step, after submitting answers to questions 3a-c, students were given the material shown in Figure 8.

At about 3PM on the day of your visit the system hangs again.

3a. What is the most logical question for you, as Amberstone, to ask the people at BU&U headquarters and at the Work Centers?

Amberstone asks, "What is so special about what happens in the work centers at 3PM?" Using the data from the computer center he sees the transactions that were requested just before the hang as his people have now instrumented the computer with special debugging data capture tools.

3b. What answer does Amberstone get, from BU&U staff, to the "most logical question" that he asked.

Figure 8 Part I

The answer is that the Dispatcher at each of the four online Work Centers asked the system for a Service Request Report at almost exactly 3PM. When you ask the Operations Manager why they all happened to request Service Request Reports at about 3PM, she replies that "we need to find out how many technicians will have to stay to do overtime work each night so that we can meet service request due dates. Dispatchers schedule overtime at about 3PM every day so that technicians can call their families to let them know if they'll be home late. The Service Request Report is the only document that tells us how much work we need to do to meet the 48 hour service commitment."

3c. What short-term actions do you take?

As a temporary expedient you ask the Operations Manger to stagger the report requests with a system administrator monitoring CPU usage to avoid a hang. You stop putting the other Work Centers online.

4a. Compare this situation to what you learned in the Overdue Book Notice Project.

4b. How could you have avoided this crisis?

Figure 8 Concluded (Part II)

As a final step, after submitting responses to questions 4a-b, students were given the material shown in Figure 9.

4a. Compare this situation to what you learned in the Overdue Book Notice Project.

The 3PM triggering of the reports is a hidden requirement just like the need to keep the Overdue Notice exactly as specified in the Overdue Book Notice Project.

Both systems are very sensitive to the business workflows or scenarios that causes bursts of certain high resource consuming transaction, such as producing the report.

4b How could you have avoided this crisis?

Get all the stakeholders, that is the VP for Customer Service, the Operations Manager, and the technicians, involved in the requirements definition process. Make sure the requirements engineers visit the manual operation sites and understand the business scenarios.

5. How would you change the architecture of the system to provide a stable solution and to avoid the hangs while honoring the need for the 3PM running of the Service Request Reports for all Work Centers?

Figure 9

After the end of the exam, students were given a copy of all questions and answers, including the answer to question 5 shown in Figure 10.

5. How would you change the architecture of the system to provide a stable solution and to avoid the hangs while honoring the need for the 3PM running of the Service Request Reports for all Work Centers?

Upgrade the server to broadcast the status of each service request to all the clients on a transaction-by-transaction basis. Upgrade the client to select and produce the reports needed by the Dispatcher whenever he needs them.

Figure 10

The point of the earlier exam was to determine what students had learned from the staffing charges made at the end of the third week. The point of Exam 2 was to determine what students had learned from the Overdue Book Notice live-through case history about requirements engineering, and, especially, about the existence of hidden requirements and the consequent need to understand an enterprise's business, at all levels, in defining requirements. Students were given full credit for any reasonable responses to questions 1 and 2 as these questions did nor involve the requirements issue and were on the exam primarily to engage students in the Service Request System case study. We will, therefore, ignore students' performance on questions 1 and 2, and will concentrate on questions 3a-c, 4a-b, and 5. We begin with question 5 and work our way backward.

Recall that students were given the real case history's answers to questions 1, 2, 3a-c, and 4a-b before question 5 was posed to them; they were given the real case history's answers to questions 1, 2, and 3a-c, before questions 4a-b were posed to them, etc., etc. Question 5 is not related to requirements engineering, but does offer some insight into the students' overall technical sophistication. Recall that question 5 is "How would you change the architecture of the system to provide a stable solution and to avoid the hangs while honoring the need for the 3PM running of the Service Request Reports for all Work Centers?"

Students had in front of them the information that "*as a temporary expedient* you ask the Operations Manger to stagger the report requests with a system administrator monitoring CPU usage to avoid a hang." Nevertheless, ten responses suggested this temporary "fix" as a permanent solution, some suggesting that it be accomplished by common agreement among dispatchers, and others that it be enforced by the software. Five responses assumed that the central system would be capable of producing all requested reports if only all other activity were curtailed ("killed") at times of high report request traffic. Seven responses ignored the request for a new "architecture" and suggested throwing additional hardware at the problem. Three students either missed the point of the question entirely or provided no relevant response.

On the bright side, eight responses can be read as proposals that the central CPU construct Service Request Reports incrementally, as transactions enter the system, thereby spreading the processing load over time and requiring only downloading of reports when they are requested. (Four responses propose this solution explicitly, and another four, though not quite as clear or explicit, can be interpreted this way.) Three responses can be read as suggesting the successful solution that the participants in the original case study arrived at, viz., distributing the computation by logging requests at the central CPU, but forwarding each, immediately upon arrival, to the relevant local site -- for local preparation of Service Request Reports on a demand basis. (Two responses propose this explicitly; an additional response can be read as proposing it.)

Thirty six students took the second exam; some provided multiple, alternative solutions to the problem, accounting for a total number of responses greater than 36. Roughly speaking, the results can be read to mean that about one third of the students were unable to come up with any real solution to the problem, about one third came up with fairly prosaic, unimaginative solutions, and somewhere between one fifth (actually, 7/36 = 19.44%) and one third (actually, 11/36 = 30.55%) of the students proposed insightful solutions.

We turn now to questions 3a-b. Recall that question 3a, together with its introduction, reads as follows: "At about 3PM on the day of your visit the system hangs again. What is the most logical question for you, as Amberstone, to ask the people at BU&U headquarters and at the Work Centers?" Fourteen students noticed that the second hang occurred at exactly the same time of day as the first, and asked what happens at 3:00PM every day -- the best possible answer. Fourteen additional students showed no evidence of having noticed the coincidence and simply asked what was going on, the day of the second hang, just before the hang -- an acceptable answer. Two additional students showed no evidence of having noticed the coincidence and asked what the peak usage hours are. Three additional students gave inappropriate answers, and three answers were unreadable on the copy of the handwritten test papers with which we were working.

Question 3b was the telling one with respect to hidden requirements. Recall that its wording was "What answer does Amberstone get, from BU&U staff, to the 'most logical question' that he asked." Of the fourteen students who realized that something was going on every day at 3:00, eight were clever enough to propose an event that

could plausibly have caused the hang. One student actually hit on what happened in the actual case history. (Three suggested that there's a shift change at 3:00PM; two suggested that employees return from lunch at 3:00PM; one suggested that the Daily Transaction Profile was run at 3:00PM; one suggested that system backup was run at 3:00PM.) Six students didn't propose plausible causes for the 3:00PM hang.

Question 4a was fairly trivial given that students had, in front of them, the fact that every dispatcher regularly asks for a Service Request Report every day at 3:00PM; it was merely a request to supply the words "hidden requirement" by analogy with what they'd seen in the Overdue Book Notice live-through case history. Recall that the wording of question 4a is "Compare this situation to what you learned in the Overdue Book Notice Project."

Twenty eight students (28/36 = 77.77%) identified the problem as a hidden requirement; we would have been sorely disappointed had the number been lower. Six missed the point entirely, and two answers were illegible. Of the twenty eight who saw that the problem was one of hidden requirements, sixteen volunteered no more than the phrase "hidden requirements." Twelve of the twenty eight showed a bit more analytical ability -- and interest in the problem -- by suggesting meaningful parallels between problems encountered in the Service Request Report case study and those encountered in the Overdue Book Notice live-through case study.

Four of the twelve drew a very apt parallel with the hidden requirement that overdue book notices must be sorted. The other eight drew a parallel was with an artifact of the original Overdue Book Notice case history which we haven't yet discussed; it wasn't quite in the category of a hidden requirement, but, rather, an anomaly engendered by Glendale School students' first experience of computer-generated, as opposed to handwritten, overdue notices. (Recall that this case history dates back to 1985, when PC's were in only a handful of homes.) In the original case history, students were so curious about the computer-generated notices that before long almost everyone in the school was taking out books and holding them until they were overdue --simply to get one of the new notices. This created more traffic than the system was designed to support, and caused it to crash; it's not quite proper to consider this a hidden requirement, partly because it was a one-time fad occurrence, but primarily because the hardware and software constraints were such that no system could have been built that would have handled the severely increased traffic. The sorting requirement was one that consultation with the appropriate stakeholder would/could have revealed [Walston1977] and was, therefore, more apt than the "everyone wants a notice" parallel, for which there existed no stakeholder who could have revealed the problem [Walston1977].

Given that 77% of the students recognized the hidden requirement [Vinter2000], how many of them understood how they might have averted it. Question 4b was directed at this issue. Recall that question 4b is "How could you have avoided this crisis? " Four students responded with "involve customers more," "communicate better with Customers," or similar, with no further elaboration; these students were given credit on the exam, but were probably simply regurgitating what they thought the instructor wanted. Six additional students suggested that members of the Software Development Team should have observed customers using the prototype or the automated system. (Note that nowhere in the case history is a prototype mentioned; students presumably mentioned it because they guessed the instructor would be impressed by their remembering the concept of a prototype as a tool in requirements engineering.) Although the six students were given credit for their answers, none of them are correct. They all failed to understand that avoidance of the problem would have required observing the *manual* process or discussing it with the end users -- before any prototyping or other implementation was done.

Eight students gave entirely wrong answers, at least one of them after having noticed that the problem was hidden requirements; one student actually suggested that the customer change his routine.

On the bright side, ten students indicated that customers should have been watched using the manual system before/during requirements/design -- a very perceptive answer, and the only action that could possibly have afforded a real chance of avoiding the problem. (In a small number of cases a bit of judgment was required to determine that students were referring to the manual system rather than to the automated system.) Eight additional students indicated that customers should have been talked to about their use of the manual system before/during requirements / design -- a second-best answer. Thus half (18/36 = 50%) of the students, admittedly with a bit of

coaching through the Service Request System case history, appear to have learned the fundamental lesson of identifying all relevant stakeholders and attempting to involve them in requirements engineering.

5. Conclusions

We begin with several general remarks regarding the way the Senior Project cum Introduction to Software Engineering course was taught. First, the use of every other class for a workshop, essentially a plenary project meeting -- not unique to the course in which the authors were involved -- both forces the teams to hold at least one serious project meeting per week, and allows the instructor to keep the teams focused and on track by making suggestions or delivering a short impromptu lecture on a relevant subject when a team appears to be in trouble. Second, students volunteered that they preferred lectures to include real cases histories illustrative of details of software engineering process and technology rather than the be concerned exclusively with "theoretical" discussions of process and technology [Benbasat1980].

As to the experiment of using a live-through case history to motivate process-averse technologists to recognize the existence of and need for software process, we begin by noting that, with the exception of the ever-hopeless two or three students, everyone got, at the very least, a small bit of what we were hoping to convey. About one third of the class achieved a moderate or better feeling for the specific process-related points of the live-through case history and of the need to take software process seriously. About one fifth of the class were profoundly influenced by the experience -- a satisfying result for an initial attempt.

As to methodology, having despaired of the use of open-ended questions, to be answered at leisure, we adopted the use of challenging questions requiring quick reasoning by analogy to assess the results of our experiment; we were reasonably satisfied with what they told us about the results of our experiment. Perhaps more, or different, information can be garnered through the use of more thoughtful, better-designed, open-ended questions; perhaps such questions would work well only if administered under test-like time constraints. We have not yet attempted to correlate responses to questions on the entry questionnaire with performance on the two exams; we are far from certain that doing so would produce interesting results as the questionnaire ended up being relatively brief. In our second experiment with live-through case histories we would like to have additional information, perhaps SAT scores and results of evaluations of students motivations.

Our plans are to develop a one- or two-semester live-through history-based course, specifically focused on software engineering practice and technology, as a prerequisite for the Senior Design course. Students will then use what they've learned in the Software Engineering course to do more sophisticated projects in Senior Design. The Software Engineering course will deal with topics like iterative/spiral development using USC's web-based MBASE (Model Based Software Engineering) tool; requirements engineering using USC's Easy WinWin software; cost-estimation using function points and USC's COCOMO II software; object-oriented architecture and design using Rational Unified Process; etc., etc.

One other university has already agreed to use our initial live-through case history in its Software Engineering/Senior Project course, and we are discussing the possibility with others. We hope to form a consortium of universities for the purpose of developing case histories relating to various aspects of software engineering process and to build a database of results of the use of the live-through method to assess its efficacy and to improve it.

Appendix

Team 2's Evaluations

Evaluation of Team 1

The product created by team 1 is obviously the most quick and efficient program of any of the groups. The program was written in C++, which, due to its compiled code, is faster than Visual Basic and Perl. Also, the program does not retrieve or store information onto the storage media except for at the load and termination of the

program. This last feature, while efficient, introduces the potential for a significant loss of data. All updated records are stored in a linked list in memory, which would disappear in the event of a power loss or system failure, while other groups wrote to the disk after each entry, not exposing themselves to this problem. Another feature of their program that was a little awkward was the way of identifying a particular book. It was strange the list can be viewed according to student name, teacher name, etc., but the lists were not sorted according to the selected criteria, and only having an index to distinguish between these seemingly unrelated items. This setup is not intuitive.

From a software engineering standpoint, team 1 seemed to have done a good job. They employed at bottom-up approach, which is something that is familiar to most projects. From our programming experience, it is easy to see large tasks in terms of the pieces that compose it. This style easily translates to software engineering and management in the sense that it is more accurate to estimate the time necessary to accomplish smaller tasks than it is for large tasks. Developing through this method is also more effective because programming on a small scale reduces some of the complexity in software development.

A thing that this group should have done differently was that everyone should have been involved. As pointed out by Prof. Bernstein, there were members of this group who did not actively contribute. This was a learning experience, and above that, it was a school project, and should be participated in by all the members of the group.

Topics that need to be taught in the remainder of this course as a result of some of the materials presented by this group are some of the testing requirements that were brought into light. Topics that were mentioned were the IEEE standards for software, and things of that nature. This material has not been covered in class, and I feel would be a vital part of the software engineering process and a necessary part of this course.

A topic from class that I feel that this group employed effectively was that of managing complexity. Visual C++ is a very complex development tool, and this group managed to create a functional program according to the specification, which is not an easy task in Visual C++. The complexity quickly grows in such a project, and it is worth noting that they were capable of maintaining manageability in such a project.

Evaluation of Team 2 (Itself)

The product that was produced by group 2 was pretty fluent. While most groups required the user to type some information about the book in order to check-in the book, this particular program did not. There was an on-screen list of books that were checked out, and the user only had to double-click on this list and the book is removed. This seems to be the most efficient way of doing things. There is nothing for the librarian to remember, nor is there a need for the user to move back and forth from the keyboard to the mouse. The user only has to operate the mouse in this situation. Also, the user interface was simple, which was a common feature among most of the programs presented, but it is still worth pointing out.

With respect to the software engineering that took place in this group, they seemed to have everything in order. There was a Gantt chart showing the milestones and time frame that it took to get this project off the ground. They claimed to have planned everything out, and seemed to have all the parts that were common to all the groups, including functional testing, stress testing, and regressing testing. As for the implementation, they did do something that was a little different. They left the programming to a few people, while the rest of the group members worked on other aspects of the engineering. This was smart because this was more an exercise in software engineering than programming, and there was quite sufficient time to accomplish this task with only a few members programming.

The thing that this group learned from this experiment was to be more prepared in the event of a change in staff. They seemed to be less adaptive than most to the change in staff midway through the project. There were members who remained on the outskirts because of this. This was a lesson to be learned.

What this group needs to learn from this course is how to manage people. As was evident from the presentation, some of the newcomers to the group did not participate in the group as much as the other members. What need to be presented in the course is how to train people who are new to the organization and how to keep people involved.

Things from this course that were particularly helpful to this group were the lessons on time and risk management. They planned out how much time was required for each of the tasks and what the risks of delays to the schedule were and how to handle problems in this schedule.

Evaluation of Team 3

The product was developed in Perl, which makes a clean text-based interface. This would be a more effective way of communicating with an advanced user, but would not be a good way to start off a user who has never seen a computer. The amount of text would make it less intuitive than the graphical interface because of the way people are. People identify more with pictures than text, and are less intimidated by it. Other than this complaint, the program was very nice. It is very quick and efficient, and over time, the librarian would learn how to work with this program more effectively, and be more efficient with this program than any others, but at the cost of having to learn UNIX/Linux.

This group had a very good engineering approach. They employed the black-box approach, which allowed some consistency between the modules. Everything was modularized so no programmer would have to worry about anything except for the particular task that they were assigned. This reduces the complexity facing most of the programmers, making it a more manageable process.

This group should have tried to develop the project on a Windows platform, because it is a more universal platform, and something that an actual school is more likely to have. Also, they should have made sure that their program was completely working before the presentation.

The thing that we need to learn in class is how to test modules. This group built modules that took care of everything, but did not go into detail as to how they tested these modules. This may be because they did not know how to perform specific tests on these modules, or that they just did not feel that this was information that was pertinent to the project. But in either case, I feel that this is a subject that should be discussed in class.

A topic that I felt that this group took advantage of was the concept of clean-room engineering. They make a good effort in making everything modularized so that nothing influenced anything else in any way. This is a concept that should be employed by more of the groups.

Evaluation of Team 4

This group has a standard interface that was created in Visual Basic. There is nothing that is particularly exciting about this program, just a solid working program. One thing that did stand out was that they incorporated a help file that was very assessable to the user through the standard F1 keystroke. This is a big plus for the program.

This group employed the prototype model in which they created a working prototype in another language just to make sure that their design was laid out. This seems to be an inefficient way of planning. Instead of going all the way to creating a prototype, they should have spent more time creating a Gantt chart and everything else that comes with the software engineering process rather than go out and produce a prototype. Developing the quick prototype would be a nice way to get things done quickly, but it is not software engineering.

Something that this group could have done differently is to create a plan within the organization earlier in the process. It seems from the presentation that their group was really in the dark as to how to approach this project before seeking help from an external source by bringing in people from another organization. While this is a

solution that accomplishes the task at hand, there should have been some kind of structure within the group ahead of time.

There are no particular topics that I feel that should be presented in class after seeing this group's presentation. They were relatively solid in all areas, leaving little to criticize.

This group used to prototype model effectively, which was something that was learned in class.

References

[March1999] Steve March," Learning from Pathfinder's Bumpy Start," Software Testing and Quality Management, September/October 1999, Volume 1, Issue 5, page 10, www.stqemagazine.com

[Lederer1995] Albert Lederer and Jayesh Prasad, "Causes of Inaccurate Software Development Cost Estimates," J. Systems Software 1005 ; 31:125-134 0164-1212/95, Elsevier Sciences, NY, NY..

[Walston1977] Walston and C.P. Felix, "A Method of Programming Measurement and Estimation," IBM System Journal Vol. 16 No. 1 1977 54-73, reprint order number G321-5045.

[Benbasat1980] Izak Benbasat and Iris Vessey, "Programmer and Analyst Time/Cost Estimation," MIS Quarterly/June 1980 pp.31-43

[Vinter2000] Otto Vinter and Soren Lauesen, Analyzing Requirements Bugs, Software Testing & Quality Engineering, Volume 2, Issue 6, November/December 2000.

[Paulk] Mark C Paulk , et al. "The Capability Maturity Model: Guidelines for Improving the Software Process, Addison-Wesley, ISBN 0-210-54664-7

[Royce1975] Winston Royce, "Software Requirements Analysis: Sizing and Costing," in Practical Strategies for Developing Large Software Systems Ellis Horowitz, editor Addison Wesley 1975 Lib of Congress 74-288818, pp. 57-61.

[DeMarco1995] Tom DeMarco, "Why Does Software Cost So Much," Dorset House Publishing NY,NY, 1995, ISBN 0-932633-34-X

[McConnell1998] Steve McConnell, "Software Project Survival Guide," Microsoft Press 1998, ISBN 1-57231-621-7.

[VanVliet2000] Hans Van Vliet, "Software Engineering second edition, John Wiley 2000, NY, NY, ISBN 0-471-97508-7.