A SOFTWARE DEVELOPER'S WORK IS NEVER DONE

John Bigelow
Boise State University

Joe Garcia
Western Washington University

Dale Rude
University of Missouri

Andrea Warfield
University of Michigan

Have you ever toyed with the idea of taking a deep dive into the third age by developing software to enhance current technologies for teaching OB? Perhaps you've been dissuaded because of lack of technical knowledge, an inability to program, you haven't a clue as to what would be involved in creating a top notch product, or you don't know how your institution would view such a project. Well, now perhaps you can put those fears and apprehensions to rest. The authors of this article, all active members of the Microcomputer Interest Group of the OBTC, share with you the realities of software development in the field of organizational behavior with in-depth descriptions of their own experiences as software developers.

Software development for teaching organizational behavior is at a very primitive stage of development at this time. Because there are no established overriding principles for such a venture, the authors offer four separate case histories of their projects as a basis for deriving principles and guidelines for future undertakings.

John Bigelow, a well-known name in both the Microcomputer Interest Group and the OBTC, begins with a very general approach by addressing various questions a novice may have regarding software development and reviewing a number of computer projects he's been involved in. He adds an extra focus to his discussion on student, faculty and administrative response to his projects.

The projects of Andrea Warfield and Dale Rude are in sharp contrast. Warfield, who had scant technical or programming knowledge when she began her project, compensated for that lack of knowledge by acquiring a piece of software called an authoring system which enables one to use the

microcomputer as an instructional tool. Warfield describes her experience in authoring a computer-based leadership training program based on Robert Quinn's Competing Values Leadership Model. Dale Rude, on the other hand, who is perhaps the most technically oriented of all four authors, programmed his own policy capturing software called the Decision Analysis System. Rude points out that there are disadvantages as well as advantages to doing your own programming!

Finally, Joe Garcia goes one step beyond software development and describes his project of developing an entire integrated package of an organizational behavior text, instructors manual, and accompanying microcomputer simulation. Garcia discusses both the dynamics of teamwork needed to create such a product as well as the new working relationship with publishers when a software product is added to a text package.

John Bigelow, Boise State University

For the last four years I have spent a significant part of my professional life developing software for use in my OB and Management classes. The purpose of this paper is to share what I have learned during this process. Others interested in engaging in software development may be interested in this narrative, since it may help them bypass some of the time-consuming trial-and-error learning which I underwent.

The process of software development might be thought of as the answering of a series of questions, something like the following:

(1) Should I invest time in software development?

(2) How can I get an idea for development?

(3) What resources can I call on in developing a program?

(4) How do students respond to software innovations?

(5) How does software development impact on my professional standing?

In the following pages I will discuss each of these questions based on my experience.

Should I Invest Time in Software Development?

I think that a number of personal and institutional factors need to be present before engaging in software development makes sense. These are as follows:

One is interested (not necessarily skilled) in the process of software development. Development of software requires a fair amount of commitment. Unless one finds this process interesting, one will probably not experience it as worth sticking to through completion. On the other hand, one need not have a lot of experience or expertise in the area. Because I find writing programs interesting, I have taken the time to obtain the necessary hardware and software and have learned to program. Others may choose to program in simpler languages or simply find someone else to do the programming work. In addition I believe that there are a lot of possibilities for

microcomputer applications in the OB/Management area, but that there are not a lot of professional software developers working in this area—and those applications that do exist tend to be pricey. By developing my own software I bypass the need for finding money for purchasing multiple copies, and create software to my own specifications.

Sufficient computer resources are at least potentially available. Many universities are still in the process of coming to grips with microcomputers and their use. It may not make a lot of sense to develop a computer application unless there are enough computers available to reasonably implement the application. On the other hand, one might elect to do some software development as a way of eliciting funds for the hardware. My work in software evolved concurrently with a university-wide drive to implement microcomputers and other new technologies into teaching. As a result there are a number of microcomputer labs available to students using my software.

There is a reasonably positive climate toward such activity in the college. I think it's important for there to be some institutional support for software development. Of course, a university is not just one thing, and one may find that different sectors (e.g., faculty, chair, dean, and president) may have differing opinions. In my setting there is mixed support. At a higher level (president, dean, and chair) involvement in developing uses for new teaching technologies is viewed quite favorably. On the other hand, faculty attitude is more ambivalent. Some faculty have little experience with microcomputers and oppose investment in microcomputers because they might draw funds from other more desirable channels. In addition, criteria for promotion and tenure still do not place much prominence on classroom innovation. The first step of promotion and tenure involves consideration by a faculty group which would quite possibly include members who may neither know much about or support the use of microcomputers, and may not place much weight on the activities in any event.

In deciding to engage in software development then, I realized that the pros and cons were mixed. As (at that point) an untenured associate professor, I realized that the impact of software development on my career was ambiguous, In the final analysis I recognized that there was some risk in spending time on software development, but that my own motivation, values, and the potential benefits outweighed the risks.

How Can I Get an Idea for Development?

My own ideas for development have sprung from a conjunction of reading about and trying others' programs and thinking about what I can do that would be educationally useful and facilitated by a computer program. In addition, my thinking about computer uses has evolved through the process of trying out computers in the classroom and inviting students to contribute their ideas.

The idea for my first program came from reading about a program which advised managers about how to carry out certain interactions. I thought, "wouldn't it be nice to have a program which led management students through some contingency theories in a structured way?" This eventually

led to a program which incorporated some management theories and allowed students to apply them via a set of questions.

I tried this program out as a basis for organizing a number of introductory management courses and found that the program was in essence too "thin" — the ideas were dealt with in their most abstract forms with little support for understanding or applying the concepts. My thinking shifted in two ways. First I began to appreciate the importance of a "rich" interface between student and computer, and second, I began to think about having students develop the advisory system instead of simply being users. I began to read about expert systems and tried out one (MicroExpert) in class. While this was one of the few expert system shells which was suitable for class use in my setting, its interface was too thin. Thus began my second project — the development of "Advisor" — an expert system shell which was easy to learn and had provisions for a "richer" interface. This project turned out rather well, and I continue to use (and refine) it as the basis for a group project in my management course.

A third project stemmed from my reading about a number of programs which simply translated existing paper and pencil instruments to a computerized form. The advantage was that it was more easily administered, and the evaluation and some interpretation could be done automatically. I began thinking about what kind of instrument would be useful in my classes. There are a number of instruments used in my organizational behavior course, but I decided it would be better to create my own, to avoid copyright problems. I finally decided on a "management readiness" preassessment instrument which would tell beginning management students about what managing entailed, where their strengths and opportunities for development lay, and what they might do to develop their management readiness. This line of thinking led to development of a "Management Readiness Tutor" which I've used as an early individual assignment in my management course, and which I think might well be required of anyone wanting to take a management major.

A fourth project evolved from my ad hoc efforts to use computer programs to manage class data. I realized that I tended to use simple grading techniques simply because it was too tedious and inaccurate to carry out more complicated grading procedures by hand. It dawned on me that if I took the trouble to enter scores into the computer, I could do anything I wanted. Recently I collected together some programs into a "Class Data Manager" which allows entry and manipulation of class information in a variety of ways. This has permitted me to do things with the data that would otherwise be too tedious to do. These include: (1) sorting student scores in various ways (e.g., by group, student alias, and class rank), (2) forming groups on the basis of FIRO scores or heterogeneity, (3) finding ways to recognize a variety of forms of student accomplishment, and (4) giving live exams scored directly on the computer.

I'm aware of a number of applications which have already been conceived of and are being worked on. These include: (1) programs which advise people about person-specific interactions, (2) interactive cases where students are provided case information in increments and the case evolves

according to their decisions, (3) group problem diagnostics and advice, and (4) expert systems which apply conceptual frameworks in the context of a situation. I'm sure there are many potential applications which have not yet been thought of.

I'm coming to realize that most of the applications I've thought of or others have thought of are in the category of taking something already being done and using a computer to do it — hopefully better. If we think of computer uses in class as a socio-technical intervention, we might also think about how computers can transform the learning process into forms not yet used or conceived of.

What Resources Can I Call on in Developing a Program?

Once the idea for a program has been germinated, it remains to translate the idea into a workable reality. The process of doing this generally will involve choosing what computer to develop and implement the software on, what software medium to work through, obtaining of institutional support, and trying out the software to see how it works.

Hardware. In general it's a good idea to develop the program using the computer on which it will be implemented. That is, if the school has IBM-PCs, develop it on an IBM-PC. In some circles the philosophy is that the program should be developed on a computer with greater capacity than the target machine, the reasoning being that the computer has to handle both the program itself and all the ancillary material such as an interpreter, compiler, etc. In my own experience it's very rarely necessary to develop a program on a computer other than the target one, and doing so creates a pretty large chore of translating the program once it's done. In my case, I develop for the Macintosh and the IBM-PC. I develop on the Macintosh (my preferred computer) but may also shift to the IBM-PC and develop for awhile too. Both are suitable developmental computers for my work.

Software. Once a machine has been chosen, it's possible to think about the medium through which software will be expressed. In my case the medium is MicroSoft BASIC and Modula-2, since both are implemented on the Macintosh and the IBM-PC. "C" is the preferred language of most professional developers, but Pascal also has a following. For those who are not interested in learning a computer language, there are a number of options. One is to use a visual flow chart-oriented language which requires little programming ability. Another may be to use an authoring system, such as Quest, IMSATT, or Genesis. Many of these systems are designed to be used by people with little or no previous experience. Finally, it's possible simply to join up with an experienced programmer who can work with one in translating ideas into programs.

Institutional Support and Time. Having established the means, the only remaining question is to establish the institutional context in which the work is done. Developing a program of any sophistication requires a fair amount of time (I have always underestimated the amount of time required to develop a piece of software. I continue to underestimate even knowing this), which needs to be established in one way or another. Sometimes it's possible to do developmental work during the summer. My school has a fund for

such work, and has been flexible in granting funds for projects rather different from those envisaged for the grant. However, I have also felt it necessary to modify my proposals so as to fit the overall parameters of the grant. This has usually meant not only developing a program, but also figuring out some basis for writing a presentable paper about the program or some aspect of it. I developed my first "advisory" program and the Management Readiness Tutor as summer grant projects.

If grant money or other support is not available, it may be possible simply to fit the project in during the semester. Since the work takes time which might be used for other scholarly activity, it might be desirable to think about not only developing the program, but also how to use it as the basis for a paper for conference presentation. I developed the Advisor and Class Data Manager during school years.

Class Trials. Once the software is in a reasonable state of readiness (realize that live software is always in a state of development) one can then think about how to try it out in class. I find that it's best to start with a tentative use of the software in conjunction with student assessments before incorporating software extensively in the class. I think this is a good idea with any piece of software, developed or purchased. In this way the program can be debugged and student reactions can be obtained before the fate of a course becomes at stake. I first implemented Advisor as an optional group project. My first use of Management Readiness Tutor was as an optional extra credit project involving using and assessing the program. Following a positive response, I made it the basis of an assigned individual paper. In my early use of Class Data Manager action exam, I kept at hand a pencil and paper version as a backup. In essence, I see the process of implementing software in the class as an incremental one, starting with light or optional use, moving to heavier use, and always with provisions for feedback and idea generation. Even after two years of regular class use, student groups are still providing me with new ideas for how to improve the system and finding new ways to crash the program.

How Do Students Respond to Software Innovations?

I have experienced two dimensions to student responses to software as a part of the class. The first dimension is a reaction to the computer itself. When I first started using computer assignments, microcomputers themselves were a novelty. Many students had never before used a microcomputer and didn't have a clue as to how to orient themselves to the machine. In those days, "computer-phobia" and "computer literacy" were terms in good currency. This aspect of computer response has changed rapidly over the last four years. Now courses are available on how to use microcomputers and microcomputers are used routinely in a number of courses. I still find a significant proportion of students who don't know where the "on" switch is or how to insert a disk, but there are usually other students nearby who can help them, and even inexperienced students seem to accept the microcomputer as a legitimate instrument of class learning. The student culture has adapted rapidly to microcomputers.

The second dimension of student response is reaction to the programs

themselves. Here I find that students often don't appreciate the program as I do. Where I think it's great to have an application which does what this program does, students seem generally matter-of-fact. Most seem to expect high quality performance, bug-free operation, and a user-friendly interface as a matter of course. If the program does what it's supposed to do in a fairly trouble-free way and students see the program's contribution as worthwhile, they'll accept the program as a valid part of the class. They generally don't seem to rate higher for all that effort in development and innovation in computer use — and they do seem to rate down if the idea or the program doesn't work as hoped.

The message here seems to be that while software innovations can contribute to learning, one should not engage in software development in hopes of raising one's student ratings. A careful implementation of innovative software in the classroom probably won't hurt one's ratings, but probably won't raise them either.

How does Software Development Impact on My Professional Standing?

As a tenure track instructor, I knew that a comitment to software development would also have an impact on my professional development. I knew that ultimately I would be assessed not only on teaching, but also on other criteria as well, such as my professional activities and publication record. Recognizing this, I have attempted to propose conference sessions based on my papers and write articles based on my programs and what I have learned about computer use in learning.

Recently, I successfully went through the promotion and tenuring process. My sense is that my work in program development was not a help at the faculty committee level. I think this was in part due to mixed faculty attitudes and in part due to formal criteria which doesn't say much about software or teaching innovation. In terms of getting a positive response at this level, I could have better spent my time working on other measures of classroom accomplishment and on more traditional forms of academic writing. On the other hand, I think that my work helped at higher levels, where development and innovation and new teaching technologies seemed to be more appreciated.

Conclusion

I have discussed the processes and issues involved in software development, and the impact of my choice to engage in it. Institutionally the results seem somewhat discouraging. It's not a way to get student ratings, it's not a substitute for other scholarly activities, and its impact on the promotion and tenure process is mixed.

At the risk of engaging in cognitive dissonance, however, I think the overall ledger balances out more on the positive side. I could have chosen to spend my time in activities calculated to score higher on the promotion and tenure scale. Had I done so, I don't think I would have the energy and enthusiasm which I currently experience for my work. I engaged in software development because it was where my interests lay and because I felt that it

had value. As a result, I have been spending time doing the things which I experience as interesting, exciting, and worthwhile. Although the activities probably did not jibe as well with promotion and tenure criteria, I did negotiate the process successfully — and have enjoyed the process much more than I would have otherwise.

Andrea Warfield, University of Michigan

The seed for my project of developing a computer-based management training module was planted in 1984 when I was traveling through England doing research on the impact of the microcomputer on English culture. I was working with writers of training programs at Ferranti, a company in Manchester, England which does computer-based training for the British military. The authors were interested in my recent experiences as a writer of Thoughtware computer based management training programs. I was explaining to them how I would design screen content in my office in Michigan, send my designs off to the programmers in Florida, and work with the programmers by phone on developing the computer modules.

One of the writers said, "You ought to get an authoring system. You could bypass Thoughtware programmers if you had an authoring system." "What's an authoring system?" I asked. "An authoring system is to a trainer or educator what a word processor is to a writer. It is a system which allows an 'author' to develop original training modules by enabling him/her to design screens in graphics or text mode, branch those screens so that they are highly interactive, and transfer the screens to a 'driver' — a computer-based training module which is easily driven by learner response. You should get one," he said persuasively. "You Yanks are always looking for the pot of gold at the end of the rainbow. So that's it, get an authoring system. You'll have your pot of gold."

Well, this was something I couldn't just walk away from and forget, so as soon as I returned to the States, I asked our librarians to research authoring systems. In 1984, we found very few companies who produced authoring systems. By 1986, the number had grown to about twenty companies throughout the United States, and I acquired an authoring system to use for one year through the Higher Education Management Institute (HEMI), a broker of software to higher education. The authoring system I have been working with over the last year is Knowledge Support System Author developed by Comware Inc. in Cincinnati, Ohio. Often developers of such software will allow educators to use their systems for course development. It's great for the educators because they learn how to author a computer based module, and it's great for the software developers because they get exposure of their system by learners completing the computer based course.

I acquired the system in April of 1986, and after three days of training, was anxious to develop my own course. The next step was to find an appropriate topic. Robert Quinn and I had been discussing the possibility of putting his leadership model on computer. After learning more about his model at the 1986 Organizational Behavior Teacher's Conference, I realized that it was the perfect content for computer based training. Now that I had

a profound technology and a profound concept, I was ready to start work.

Launching the Project

My original plan was to do all the work myself. Now that I look back, I realize how dramatically unrealistic that was. It is impossible for one person to develop a good computer-based management training program because you need so many different kinds of expertise. Fortunately, enough people were interested in working on the project that a team quickly formed to launch the program.

My role was the project director. I have very little technical expertise, and the talents I needed for the project were the capacity to bring together people with a wide variety of abilities and interests, keep channels of communication open so that we could all understand the perspective of each member of the team, and keep the team motivated in pioneering this project. There were two programmers on the team, both of whom had excellent technical expertise: Bill Rush, electrical engineer at Ferris State University and Carol Gustafsen from Michigan State University. David Hessler, Professor of Instructional Design at University of Michigan, oversaw the instructional design of the module. Richard Kline, faculty member at the art department of Central Michigan University was the art director. Clark DeHaven, Director of Executive Training Programs at Michigan State University, served as director of content design, the person responsible for ensuring that the module was effective in management training. Christine Barajas, Kathy Peterson, and Cheryl Tinskey were students who took independent studies and did the actual authoring.

Process

Our first step was to organize the entire project by breaking it into logical components and setting bench marks for the completion of significant tasks. The first major task was to redesign the content of Quinn's manuscript from a print-based to a computer-based format; the next was to do the actual authoring; and the final challenge was to test the module on management trainees.

In redesigning the content from print to computer-based material, we added an important new dimension: the use of color to indicate various sections of the training program. One of the advantages of computer over print-based training is that engages many of the senses. The use of color not only enlivens the material but serves as new symbols for important concepts. Using color as an organizing tool, we transferred the print based material to story boards—boards containing pictures of a series of screens which were individually designed to display the appropriate material from the manuscript.

This first, and very significant, task set a strong foundation for the actual authoring of the course. Students worked with the story boards and manuscript and authored the actual computer screens. We met periodically to review the designs and make any changes that were necessary. While the design of the story boards took approximately two months, the authoring

took about eight months. We completed 620 screens and are currently working on more.

It is very important to note that a significant portion of the module was done in BASIC or Pascal programs and integrated into the course. An important quality of computer-based training is that it is interactive. When the authoring system did not allow for as much interaction as we wanted in a particular section, we programmed that section and integrated it into the program. Another use of programmed sections was for instrumentation. A dramatic example of the superiority of computer- over print-based management training is the quick scoring of instruments which are laborious and time consuming when done by hand. Pascal was used for instruments which required complex scoring of instruments.

I wish I could say that the story ended happily with the module testing beautifully with the management trainees and that I had indeed found my pot of gold at the end of the rainbow. But that is not the case...yet. I, like John Bigelow, underestimate the time it takes to complete a software project even when I know that I tend to underestimate it. At this writing, about 80 percent of the course is done. The remaining 20 percent will consist of the more interactive exercises and tests throughout the course to assess the learner's progress. And that, of course, will simply be the first edition. Software is constantly being refined and updated. It is literally never done.

Resources

The greatest resource needed for the development of good software is talented people with a commitment to see the project through. The determination to create a product from the ground up with little guidance in dealing with the countless intricacies which inevitably occur call for characteristics such as zeal, perseverance, tenacity and patience. Because of the pathbreaking nature of such an attempt, clear communication is vital among team members. In addition, computers, a good authoring system, and at least one team member with technical expertise is needed.

Impact

The development of computer training modules like these offers new and exciting challenges to teachers of organizational behavior and innovative learning opportunities for students. To design the content of an effective computerized training module, the teacher must work with new assumptions regarding instructional design. For example, classroom instruction is linear in that a course is launched at a starting point at the beginning of a term, proceeds in a logical fashion, and is completed at a concluding point when the term has ended. Computer instruction does not follow this process. Instead, course content can be accessed at any point and is highly individualized as to how much time the learner spends in any one section.

Content presentation on a microcomputer is not as straightforward as in a text. To avoid the biggest criticism of computer-assisted learning, which is that it is simply page turning, screens must be designed with movement, depth and precise timing. This calls for talents and skills which are closer to

those of a director than a writer.

The use of color, movement, sound, interactive exercises, immediate scoring, and feedback creates a vivid multidimensional experience for the student. A well designed computer-based course will engage all of the senses of the learner.

In short, the potential for using the microcomputer as an instructional tool for the teaching of Organizational Behavior as well as other disciplines and fields is exciting, challenging and limitless.

Dale Rude, University of Missouri

The Decision Analysis System is a collection of programs which support policy capturing (Slovic & Lichtenstein, 1971). Policy capturing is a regression-based approach to modeling human judgement. Instead of asking decision makers how they make judgements, judgements are collected from each decision maker and regressed upon key decision variables. The decision makers receive feedback concerning (1) how consistent they were in applying their decision strategies, (2) which decision variables they used, (3) in what manner the decision variables were used (direct, inverse, etc.) and (4) the relative importance of each decision variable.[1]

Inception and Launch of the Project

As a student in a managerial information processing class, I was impressed with the power and utility of the lens model theoretical framework and its policy capturing methodology. As I talked with my fellow students, I learned that many of them neither fully comprehended nor appreciated policy capturing and the lens model. After musing over this, I decided that, for most students, experiencing a policy capturing application was necessary for understanding the lens model and policy capturing and that once understood, appreciation might then follow. Thus, I began looking for a computer program which would enable students to experience policy capturing by entering judgements and receiving detailed individualized feedback.

My search revealed that a FORTRAN program called Policy had been developed by Kenneth Hammond and his associates at the University of Colorado. Available through a time sharing computer service for $16.00 per hour, it was too expensive to be practical for class use. After searching in vain for other programs, I began development of my own program.

Process

Development of the Decision Analysis System has proceeded in three separate phases and has resulted in three distinct versions of the programs. Phase I began shortly after the completion of the class, required about two months of development time, and produced a basic judgement collection program and a simple SAS batch program for analyzing judgements and providing individualized feedback. Students entered their judgements into an HP1000 computer and received feedback consisting of a correlation

matrix (containing the correlation between their judgements and the criterion) and a regression equation resulting from the regression of their judgements onto the decision variables which they had been given. The task consisted of estimating the change in CEO salary using data concerning the CEO's previous salary and company's performance. The need for a new task and revised feedback became clear when my students complained about the statistical feedback and the impossibility of making accurate judgements concerning a criterion which they knew little or nothing about.

Phase II began shortly after I started a new position at the University of Missouri. I invited one of my new colleagues to join me in this development effort. We chose a new language, interactive SAS, but retained the CEO change in salary task. Eight months later it was operational. The program would collect judgements and produce immediate feedback; however, the cost was about $25 per person, an impractical expense. Thus, it was time to begin anew with Phase III.

By now, IBM PCs were available and were chosen to be the computing machinery for Phase III. APL was chosen for the programming language and proved to be a very useful and powerful tool. After four more years of development Phase III and the Decision Analysis System are now complete. An inexperienced user can easily create the decision environment of choice, collect judgements from decision makers and provide readily understood feedback.

The major resources required were time, computing equipment, and software. Over two person-years (at least one of which is mine) have been expended in development of the software.

Unanticipated Issues and Problems

Three unanticipated issues and problems—the seductiveness of programming, the cost of licensing, and the time and effort required for preparing the system for others' use and documenting the system—have complicated development of the Decision Analysis System.

The seductiveness of programming was a problem. In Phase III, I performed over two thirds of the programming myself—the rest being done by my research assistants. My devotion to programming delayed other important activities such as preparing my dissertation for submission to a journal. I find programming to be intrinsically rewarding because of its structure, quick feedback regarding success or failure, and its challenging yet solvable problems. In addition, my commitment to the project rose as my investment in it increased. Thus, if entertained by programming as I am, it's important to carefully control one's programming involvement or to find someone else to do it for you.

The cost of licensing should be an important decision variable in the choice of a development language. By the time we considered licensing our software for distribution, we were struck with APL because of our heavy investment in the programmed code. Unlike many other languages which can be distributed in compiled form for free and without registering each application with the software company, each application of APL which we distribute must be registered and a fee paid for it. Two fee options are

currently available: (1) $100 per application, or (2) 10 percent of net sales after payment of a nonrefundable $2,000 deposit.

A third unexpected issue has been the time and effort required for writing documentation and for transforming a working program held together by baling wire into one an unfamiliar person can easily use. Every input step must be carefully explained and completely error trapped. All procedures and the underlying logic and structure behind each step should be clearly and unambiguously described.

The Impact

The Decision Analysis System is a very useful tool for teaching and research. It is a superb teaching exercise which provides the experiential introduction to the lens model and policy capturing which I had sought from the beginning. Typically the class sessions in which we process policy capturing results are the best of the semester.

The System is a very useful tool for researching decision making. One can easily devise decision environments and collect data. The quick and easily understood feedback makes it easy to study the effects of policy capturing feedback upon decision makers.

Personal benefits to me from development include an in-depth understanding of policy capturing and many ideas for future research projects. Transforming a research methodology into a software tool for teaching and research requires complete understanding of the procedure and the questioning of existing findings and perspectives. The process has revealed inadequacies of the policy capturing procedure for future inquiry and methodological development.

Joe Garcia, Western Washington University

In this section, I will describe my experience in collaborating and managing a project to develop microcomputer-based simulation for integration with an organizational behavior text (Garcia & Lewis, 1986; Garcia, Lewis & Fiedler, 1986). Let it be known from the beginning that I am not and never expect to be a computer programmer. Let it also be known that an understanding of organizational behavior and what our team wanted as a final product for the classroom was critical to the success of the project. And finally, let it be known that having a computer programmer who is familiar and comfortable with instructional design and classroom requirements is a great asset.

Inception

The purpose of our project was to attempt to marry experiential learning with knowledge of organizational behavior research content. To this end we designed a product which was divided into two primary components: The first was composed of five chapters on topics important to organizational behavior, i.e., motivation, communication, decision making, leadership and training and development. The second was a microcomputer-based

simulation of a mining organization with a four quarter life in a competitive futuristic setting. This simulation requires student teams to make management decisions based on the chapter topics and significant quantitative business concerns, i.e., pay, bonus, training expenditures, etc.

The role of the computer in the simulation is one of support in the form of processing data and generating feedback to students and the instructor. Students never directly interact with the computer. They simple see computer reports of various types. Instead, the instructor or teaching assistant uses the microcomputer to set up the simulation, input data, and generate feedback.

The initiative for the project came from Chad Lewis who had worked with Phil Lewis, a computer programmer and educational consultant (Lewis, Lewis & Gale, 1985; Lewis & Lewis, 1984; Lewis, 1983) in successfully developing microcomputer simulations for general business and marketing courses. The Lewis' expertise in software design and development literally got our program rolling. We began with a proposal for a product which would integrate software with student and instructor paperware for Introduction to Organizational Behavior class materials. The proposal laid out detailed responsibilities for writing, coding, and testing the product. We then contacted several book publishers to find if there was any interest in supporting our proposal. We were pleased to find positive responses, and we finally signed with Allyn & Bacon, Inc.

Launching the Project

Beginning a project such as ours required that we immediately address three important issues: (1) product quality, (2) marketing strategy, and (3) support for resources required to complete the project.

Our design emphasized quality by stressing the link between student learning in the text with the software-based simulation. We created these links by including features such as tying student feedback reports to page numbers in the student manual so that simulation performance feedback was specific and easy to obtain. To ensure face validity, we built an algorithm which allowed student teams who performed best on the management decisions, reflecting knowledge of subject matter and good business decisions, to generate the highest profits.

Our intended audience was students in introduction to organizational behavior courses and their instructors. After examining our own abilities to distribute and market our product, it became obvious that we needed help. We had no financial support or direct way of distributing our product to teachers of OB. Our only viable option, as we perceived it, was to approach the textbook publishing industry. After all, we felt, why reinvent the wheel? The industry has an active marketing and sales force which could help us reach our market.

The issue of support was in part addressed by our solution to the marketing issue. Once again we turned to the publishing industry, which is motivated by the opportunity to sell products and make a profit. We were fortunate to find interest from several publishers to share in the risk of supporting and developing the product. Our success in obtaining support was

due to three factors: (1) our record of completed projects, (2) the concept of building an integrated package, and (3) an opportunity for a publisher to be among the first to move microcomputer technology into OB textbook offerings.

The Process

We developed our product in parallel fashion: The major responsibility for writing the organizational behavior topic materials and instructor's manual fell on my shoulders; Chad Lewis was primarily responsible for developing the simulation materials; and Fred Fiedler served as our conscience and editor. We hired Phil Lewis as our programmer, first on a contract basis and later as partner on the team.

The software was written in BASIC for IBM and Apple II microcomputers since they are the two most widely available in education settings. The actual software development of subroutines and other technical details was the sole responsibility of the programmer. We simply specified to him our needs in terms of inputs, outputs and our algorithm as we envisioned it. As the program developed, team members tested the algorithm output to be sure it was generating the patterns of data we had in mind. The screen designs were generated by the programmer such that all screens which had a physical referent, (e.g., student decision forms and team management reports) mirrored the referent. Designing the user-friendly instructions included taking the screen dumps and arranging them in the sequence that a typical user would use the program, thus creating a "walk" through the program guided by the screens with explanations in the instructor's manual. The program itself is menu driven.

Another important activity was the creation of a scenario bank or set of management situations for the simulation that drew on the text material. As with many test generation programs, our scenarios had to meet the length restrictions determined by the amount of memory available on diskette and RAM. Because the scenarios would be tied to the student manual, we could not begin generating scenarios until the topic section was already or nearly written. An important consequence of the text and software integration was that software completion was delayed until the pages were determined for the student manual. This was especially critical in scheduling production because page numbers from the student manual had to be assigned to the items in the scenario bank. As mentioned above, these management feedback reports would indicate to students where an explanation of the rationale for a scenario could be found in the student manual.

Testing the software and the simulation were two separate activities. We ran the program numerous times to find out if there were keystroke sequences that would crash the program. Copies of the software were then sent to the publisher to be tested and were also informally tested in-house. Unfortunately, we never had a full beta test (i.e., a test conducted by a disinterested party) of the software. I attribute our failure to have a beta test to the relative inexperience of the publisher with software development and our assumption that the publisher would be familiar with test procedures. Fortunately, our fears about hidden bugs in the program were never realized.

We conducted classroom tests of the simulation as the page proofs for the manuals were being reviewed. Using duplicates of the manuscript and nearly completed copies of the program disk, our students helped us learn more about the simulation and how to use it in the classroom. Our experience in the classroom led us to the conclusion that there were no fatal errors in the materials. It is regrettable, however, that we were so far into the production process that the fine tuning of the written materials could be made only for the instructor's manual.

Resources

Like many other projects which have time and quality constraints, we found that maintaining good communication between the team members and our editor and hard work were our most valuable resources. The collaborative nature of the project necessitated frequent communication. In concrete terms, phone bills, postage and other incidentals were more than I care to remember; however, we were able to obtain support for these activities as part of our role as teachers and course developers.

To support our equipment and time, we were able to obtain a fairly large advance from our publisher. Without this support, we would have been unable to pay for our programmer's time, purchase a utilities program to enhance the simulation program, and contract out two of the chapters in the student manual to free ourselves to devote more time to the integration of the overall project. This advance, of course, barely covered the authors' time, but that is often true of textbook writing as well.

In terms of equipment, we all owned compatible microcomputers. This speeded up our revision and editing process. Our programmer had both an Apple and IBM to use for creating the two versions of the simulation. At our institutions, both types of computers were available for testing the simulation which was particularly helpful.

Our scarcest resource was time. We underestimated the length of time it would take to complete the project. Weaving together the software, student manual and instructor's manual was a detailed and time consuming process. While we met our project milestones and completed the project according to schedule, it was very trying. A conservative estimate is that the project required 2 person-years.

Impact of the Project

The impact of our project will be determined by how it will be recognized by our peers, publishers, and students. I feel one of our major successes is that it is viewed as a textbook with an integrated computer-assisted simulation. This is important for us in that we hoped to enhance, not replace, traditional teaching technologies with the microcomputer.

A nontrivial benefit of the textbook view of the product is that producing a textbook is understood by tenure and promotion committees. Lack of any formal review process is a problem these committees have in judging instructional software produced independently by an individual. The integration of software with a published text overcomes this problem. In this case,

university tenure and promotion committees can rest assured that since the project was published by a textbook company, it is subject to the usual review and scrutiny accorded any other product backed by that publisher. While this process is not equivalent to reviews conducted for scholarly journals, it has recognizable merit, especially in universities with a strong commitment to teaching.

There were other effects of our decision to work with a publisher. Ownership of the product was handled as authorship of a book. While it may not have had the most optimal results in terms of personal profits, it was a practical, straightforward and predictable way to handle ownership. For example, the licensing and use of utility programs in our product was covered by the advance we received form the publisher as well as permission to use copyrighted materials in a text. As in textbook publishing, the marketing and distribution of the product remained under the practical control of the publisher and the sales force. Our interest in proper marketing and protection from pirating was promoted through the sharing of risk with the publisher (e.g., compensation through sales royalties).

We did discover that, at least when we began our project, many publishers were naive regarding software development procedures, except perhaps for computerized test banks. For example, initially they had a poor understanding of the technical quality requirements for releasing software. As a result, we all have developed a better appreciation of the development cycle of integrated learning products.

Another important lesson for us was that this type of project required more time than anticipated because each of the components had to be linked together. In scheduling this project, we would have been well served with a separate category called integration. For example, we clearly did not anticipate how much time our programmer would spend adding code at the end of the project.

A concluding point is that we were able to demonstrate that subject matter experts, teachers of organizational behavior, can successfully collaborate with computer specialists to create instructional software for OB. By having a clear idea of what we wanted to accomplish, and being willing to work with our technical expert, we were able to translate our concepts into reality. I hope that this experience encourages some of you to try out your ideas on how computers can improve teaching organizational behavior in spite of any inhibitions you may have due to lack of interest in or knowledge of computer programming.

Conclusion

It is clear that you don't have to be a "techy" to develop good OB software. However, you definitely must have a technical expert close at hand and ideally on your project team. Instead, the challenging problems in software development are human as well as technical. How do you keep channels of communication open among team members? How do you express OB concepts to technical people and vice versa? How do you keep team members motivated in spite of the countless obstacles and intricacies of

software development? What is the response of organizational members to your work, i.e., students, faculty, administrators? Who is the audience for your product? And even though technical equipment is an essential resource, the bottom line basic resource which all authors emphasized was time. All authors seem to warn that, even though they've told you how much time software development takes, you'll still underestimate your time...you'll never beat the clock by developing software.

Note

[1]The Selection Decision Exercise is a sample application which is very useful for teaching about the lens model and policy capturing and is available from the author at no charge.

References

Garcia, J.E., & Lewis, C.T. (1986). *Instructor's manual for: People, management, and productivity.* Newton, MA: Allyn & Bacon, Inc.

Garcia, J.E., Lewis, C.T., & Fiedler, F.E. (1986). *People, management and productivity.* Newton, MA: Allyn & Bacon, Inc.

Lewis, C.T., & Lewis, P.C. (1983). *Enterprise sandwich shops.* New York, NY: McGraw-Hill, Inc.

Lewis, C.T., & Lewis, P.C. (1985). *Marketing peanut butter.* New York, NY: McGraw-Hill, Inc.

Slovic, P., & Lichtenstein, S. Comparison of bayesian and regression approaches to the study of information processing in judgement. *Organizational Behavior and Human Performance, 6,* 649-744.