RECORD OF SOCIETY OF ACTUARIES
1995 VOL. 21 NO. 3B

WHY SYSTEMS ARE ALWAYS LATE

Instructor: RANDALL ALEXANDER KAYE

Why do systems projects seem to take so much longer than originally thought? If we spend the extra time, do we get a better system in the end? Why does testing take more and more time, and seem to be less and less effective? We have all these project management tools; why do we still have problems managing these projects? Systems development is a technical discipline different from actuarial work in both methodology and expectation. As actuaries become more and more involved in systems activities, we must better understand these differences.

MR. RANDALL ALEXANDER KAYE: I am a member of the Computer Science Section council, and I am also the Computer Science Section's representative for the spring program committee.

My background has been primarily in the area of life insurance, first with small Canadian companies in an actuarial capacity. Then by working on the reserve valuation, I became more acquainted with administrative systems, eventually spending all of my time on system-related issues, even though I was still positioned in the actuarial department. From there I worked for a U.S. software vendor for several years, then returning to a life insurance company environment where I became director of information systems. Finally, I have been providing systems consulting to LOGISIL's clients for the last six years.

It's not an easy mix—actuarial and systems! A vice president of systems for a reinsurance company once told me that it would not be easy developing this niche, and he was right. As he explained, systems people will never see eye-to-eye with actuaries, since in most other industries, it's the systems people who are the prima donnas!

OVERVIEW

In describing why systems are always late, I want to start at a high level, examining North American management, and then become more detailed, discussing systems management, how users have become involved in the development of systems, then turning to the formal systems development process. Next we'll continue by seeing the challenges facing project managers of systems projects, as they try to do in-house development. Since this is so difficult, many companies turn to outsourcing and vendors, although, as we will see, these have their own problems.

NORTH AMERICAN MANAGEMENT

North American management people are too preoccupied with the "quick fix" or "magic bullet" that will solve all of their problems immediately. They want to apply technology willy-nilly, without trying to find out the root cause of the problems. Technology is just an amplifier, so applying technology often serves only to "turbocharge" the bottlenecks, and can bring a badly managed company to its knees.

Too often, I find the disappointment executives feel with systems projects, and with what technology can do for them, caused by their expectations being inflated out of proportion. The suppliers of technology, both in-house and external, are at fault by not managing their "client's" expectations properly.

Another thing you may not realize, but I've found is vice presidents of companies exaggerate. They exaggerate to their counterparts on the golf course, and they exaggerate in articles in the trade press. In so many areas where I've had firsthand experience, I know that these projects didn't go as smoothly as they say, and the changes their company experienced haven't been as beneficial or far-reaching as they claim they were. If they were, why do we see them so soon looking for another technological fix for the same problems? Perhaps it's their egos talking, but I'd much rather talk to a programmer than a vice president to find out how a project progressed or how good a new piece of software is.

The Japanese don't go in for the "quick fix;" instead they embrace *Kaizen*, or continuous improvement in incremental steps. The Japanese are looked upon as the best engineers, although they themselves recognize they aren't good at software development or industrial design. For example, the Mazda Miata was designed in California, and U.S. companies are contracted extensively to supply software.

REENGINEERING

The Japanese aren't looking for radical change as we are, with all of our reengineering projects. Today, everything seems to be a reengineering project. Too often our executives are just paying lip service to the latest management buzzword, as a means of attempting another "quick fix."

Michael Hammer defined business process reengineering (BPR) as the fundamental rethinking and radical redesign of an entire "business system" to achieve dramatic improvements in critical measures of performance. Its key concepts are that business rules were created before the availability of computer systems (and are therefore paper-based), that 80% of the administrative processes add no value (since most effort manages work flow or verifies previous work, such as duplicated data entry into various systems), and that the application of information technology enables entirely new processes. It is estimated that 50–70% of BPR efforts fail, due to either not getting the business strategy right first, or being ham-strung by middle management who feel the most threatened and have the most to lose, or by change that isn't radical enough. Successful BPR projects recognize and deal with the resulting radical changes in job descriptions, the reward structure, and the ways of measuring performance.

Many systems projects are being misclassified. Reengineering often gets confused with downscaling, since executives use common technologies, but they are very different things. Reengineering projects are driven by the company's business areas, while downscaling projects are driven by the information systems department. Reengineering's cost/benefit analysis derives its benefits from lower corporate operating costs (that is, a lower head count), while downscaling derives its benefits from lower information systems costs. In successful reengineering projects, systems costs actually increase! Successful companies pick one or the other. Too many times, executives read two different articles, one lowering technology costs, and one lowering corporate costs and they think they can do both; they can't. Is North American management leaning to centralize or decentralize? I've heard of a consulting firm with two phone numbers, one for clients who want to centralize and another for clients who want to decentralize. This isn't as far out as it sounds. I know a CEO of a life insurance company who was asked by his staff if he favors centralizing or decentralizing, and he said "Both!" He believes in centralizing for a

few years, then in decentralizing for a few years, not because he changes his mind; instead, he uses this as a catalyst for changing his company, and for reevaluating itself.

LESSONS

How should North American management invest its dollars? Surprisingly, a General Motors study showed no correlation between investing in technology and improvements in productivity and quality. However, there was a correlation when investments were made in human resources reform, such as reorganizing into teams, empowerment programs, education, and training.

The lesson we can learn from this is how important "people" technology is; how we must continue to lower the barriers of communication between people and computers. We can see this stress on the human interface, in all of the development today in graphical user interfaces. Do you remember when people took programming courses because they were worried the computer was passing them by? Well, it actually turned out that the computer had to come to them and become more accessible and easier to use. Another area of development illustrating how technology facilitates human communication is the work on local area networks (LANs) and the new hot topic "groupware," bringing people together.

The real lesson of reengineering is to reorganize and simplify the business processes before applying technology.

Another lesson for us is to beware emerging technology. Don't think you have to be first to use new technology. Instead, you should be creative and innovative in how you apply traditional technology. Only large corporations can afford to speculate or "gamble" on new technology. Victor Janulaitis, president of Positive Support Review, recently noted that only 5–8% of companies today are leaders in the competitive use of technology: "These are mostly small entrepreneurial risk takers and large corporate R&D departments. Everyone else seems content to pursue a protective follower strategy." As a CEO of a medium-sized office equipment supplier said, "We don't want to die on the cutting edge of technology. We'll wait until new technologies are proven, then use our sales and services organization to get and hold market share."

SYSTEMS MANAGEMENT

Now let's turn to the management of systems departments. One person I talked to thinks that to best explain modern systems management, one should turn to chaos theory. Seriously though, we have seen the rise of a new corporate title to accompany CEO, chief operating officer (COO), and chief financial officer (CFO); that is, the chief information officer (CIO). In most companies this is the vice-president of information systems (IS).

But systems departments are criticized for being slow and unresponsive; users want more control. The CIO seems most concerned with maintaining levels of procedures and control. Perhaps the IS department has become too big. I recently attended a presentation on human resource issues regarding acquisitions and mergers. At that company, they use rules of thumb to model the resulting merged company's organization. I was surprised when the suggested size of the IS department for a 1,000-person company was about 200! I think back enviously of how much we could have accomplished in some of the much smaller companies I worked in, if we could have had 20% of the total staff devoted to information technology.

I also think that all of the emphasis on procedures and control may also be based on the CIO's fear of having the big mistake made on his watch. Indeed, statistics show that the annual turnover rate among CIOs is about 25%. No wonder some people say that "CIO" stands for "Career Is Over."

The traditional IS department was structured along process lines, with separate departments for systems programming, database administration, network management, and application programming. Today, the CIO brings himself closer to the core of the business by organizing by application area, with separate departments for sales applications, engineering applications, marketing applications, and so on, with one department to provide technical support.

SURVEY OF CIOs

Deloitte and Touche publishes an annual survey of CIOs. The questions change each year, so you can't necessarily compare the responses from year to year. Some of the more interesting items I noticed from one of the recent surveys, is that it seems all projects are called reengineering projects. Another thing I noticed is that when companies outsource, they usually only outsource the data entry, data center operations, and network management functions, not the entire IS function. Generally, the expected cost savings from outsourcing are not realized.

But what I wanted to bring to your attention from this survey are the following two lists. The first list shows what CIOs think are the most important reasons for systems development cost overruns. In order, they are:

1. Increased scope of development
2. Lack of user consensus
3. Inaccurate project plans
4. Lack of user involvement
5. Inaccurate cost projections
6. Lack of qualified personnel
7. Inaccurate processor estimates
8. Cost of training
9. Staff turnover

Number one is what I call *creeping scope*, which is the tendency for just a little bit more additional functionality to be added to an existing project, over and over again, without adjusting project schedules and expectations accordingly; we'll cover creeping scope in more detail later. The interesting thing about this list is that three of the top four items can be remedied with more communication, understanding, and relationship building between users and systems developers.

The second list shows which application development techniques CIOs say they are using most. In order, they are:

1. System development methodologies
2. Fourth generation languages
3. Personal computer (PC)-based development
4. Prototyping
5. Joint application development (JAD) sessions
6. Computer-aided software engineering (CASE)
7. Application generators

8. Rapid application development
9. Object-oriented programming languages

Number five, JAD, is a technique in which users and systems developers jointly design and plan a system (and its development project) through highly structured, facilitated workshops. The curious thing to notice here is that JAD is the only technique listed that involves users. And even then it's number five on the list. It seems that CIOs still aren't getting their own message.

SYSTEMS DEVELOPMENT WITHOUT USER INVOLVEMENT

I like to show a series of cartoons entitled "systems development without user involvement," each showing variations on a theme (a child's swing, attached to a tree branch), but with different captions:

1. "As Proposed by the Project Sponsor" has two ropes to the swing but three boards several inches apart.
2. "As Specified in the Project Request" has three ropes and one board.
3. "As Designed by the Senior Analyst" shows two ropes and one board, but the ropes are attached to the tree's trunk, rather than the branch.
4. "As Produced by the Programmers" has two ropes and one board, but the ropes are tied to different branches on opposite sides of the tree.
5. "As Installed at the User's Site" shows the previous picture, but with the entire tree propped up on huge crutches.
6. "What the User Wanted" shows a spare tire hanging from the branch by one rope!

Is it any wonder that users are dissatisfied? Attempts have been made to try to involve users more, for example, in facilitated workshops such as the JAD sessions. Another method has been to try to reorganize, either by centralizing (moving users into the IS department) or by decentralizing (moving IS staff out to the user departments). I'd like to recount a portion of Smitty Grayson's experiences at Western Farm Bureau Life Insurance Company. In "Battling the Backlog Beast," (*The Interpreter*, Insurance Accounting Systems Association (IASA), September, 1985) he writes:

> A department entitled Methods and Procedures reporting to the Senior Vice President of Operations was formed with the purpose of developing application program specifications, providing user education, and testing all new application systems. The user departments were considered exempt from testing activities and allowed to concentrate on normal day-to-day production work. The user areas did not experience the same rate of progress as that of the Methods and Procedures department. The knowledge of new systems possessed by the users was insufficient to properly support their introduction. The demand for assistance from the Methods and Procedures department was overwhelming and since they had little authority over the areas in which they were obligated to serve, the attempt proved unsuccessful. A new group arose from the ashes of the old Methods and Procedures area that was recognized as a working unit within the systems division. Upon its inception, it was dubbed the Quality Assurance (QA) department. The objective of the department was to perform the actual testing of code from programming. The emphasis on providing user education and specification development by individuals of this area was relaxed. Unfortunately, the priorities of the QA area frequently conflicted with those of the user departments and, all too often, the unit testing required more specialized individuals than those in the QA

department. Because of the dependency of the user departments on the QA personnel, this endeavor also failed and the group was ultimately disbanded.

After lengthy debate by the different areas of the company regarding this dilemma, several conclusions were reached:

1. The education of personnel and testing of specific changes should be performed by highly specialized individuals, and they should be directly assigned to the managers of the department they intend to serve.
2. The data processing area should take a position of support in matters of research and development rather than assume control.
3. The feasibility, justification, and specification phases of a project should be performed by the department requesting the change.

These requirements laid the foundation for the evolution of a position known as User Analyst. The former employees of the QA department were reassigned to specific user areas based upon their education and experience. Since they report directly to the manager of their newly assigned areas, they essentially disassociated themselves from the systems division. The requirements of specification development, educational instruction, and departmental proceduralizing were displaced to the User Analysts leaving the programming department to perform pure data processing functions."

User departments have also become more involved by hiring and developing "power" users, extremely computer-literate user staff. Typically, the IS department has tried to ignore "power" users, since they tend to oversell the benefits of new technology. Understandably, "power" users feel disenfranchised by the IS department, and say IS people will stifle innovation when they bring in their rigor and discipline. Some "power" users are "cowboys," and insist upon their freedom from standards and procedures.

Excellently managed companies find a beneficial use for "power" users, that of an early warning system for technologies that will help the business. These companies foster an environment where "power" users are encouraged to raise new ideas, but the ideas must be far-reaching to many facets of the company's operations. "Power" users are brought into the usability lab to supervise the evaluation.

SYSTEMS DEVELOPMENT PROCESS

Now, let's talk about the formal systems development process. You've probably all experienced these steps:

1. Excitement
2. Frustration
3. Disenchantment
4. Search for the guilty
5. Punishment of the innocent
6. Distinction for the uninvolved

Seriously though, the systems development process is a rigid and formal process, beginning with the request from a user, through analysis, design, programming and testing, finally putting the system into production. In this scenario, users seem to be asking, "What can you do for us?" and the systems people are asking, "What do you want?"

As we discussed before, this process is prone to creeping scope, where little by little, more and more additional function is added to the scope of a systems project, without adjusting the project schedules, until finally a project becomes too large and unwieldy. For those fringe users who join in and add more ideas as to what the system should do, the stakes are lower in making the project successful, even though they are expected to shoulder their part of the burden.

To me, the major problem I see in the traditional management of systems projects, is that the expectations of the participants are not managed properly. To get a project approved, it seems sometimes as if it must be oversold. Users and executives become overly and unrealistically optimistic of all of the great benefits the system will provide. This can lead to having no alternative plan when things go wrong, so that eventually good money may be thrown after bad in a futile attempt to maintain a heightened positive approach. And traditionally, systems people have been bad communicators.

Some projects also flounder due to a lack of clear authority. Traditionally, systems people have been completely concerned with the technical aspects, and are ignorant of the business; and users have been completely concerned with the business, and are ignorant of software concerns. After several projects flounder without clear leadership, long-time employees can become cynical.

Some projects are just too technical to begin with. The systems people want to use the latest technical tricks to prove it can be done. In the end, these systems often need to be rewritten, or if you prefer, to reinvent their wheel, before they can become productive.

As users and executives we must realize that this isn't an exact science; because of the complexity inherent in today's systems, these are not routine projects. When what you are attempting to do hasn't been done before, it is very difficult to estimate the work, time, and money required to complete the job.

REACTIONS

So how have we reacted to better manage systems development? First, every project has a user sponsor, with user involvement throughout, and especially at the project executive level.

Second, we've seen the rise of integrated systems to address productivity issues. Again, as Smitty Grayson says:

> In the past, software packages were acquired in order to satisfy an immediate product need. For the larger companies this is probably the most practical approach; however, it causes significant problems for the medium- and smaller-sized companies. An inordinate amount of time and effort is exhausted trying to support multiple systems. The word *interface* mysteriously began to appear at all levels of communication. A serious division of resources was experienced throughout the company, but it was no more prevalent than within data processing. The solution to this problem was to consolidate these different processing systems into a single entity. Only then would we be able to redirect our resources from the interface bottleneck to more desirable enhancement and refinement activities.

Integrated systems are not a series of subsystems, but one integrated whole, with a consistent design philosophy in its user interface, technical programming style, and data

structures. This enables companies to leverage their staff's training so that, when a user or programmer approaches another part of the system, he or she can count on the same, familiar concepts being used. Rather than buying a new subsystem from another supplier, this philosophy encourages continual improvements within a consistent framework. Hmmm, sounds like "Kaizen!"

One of the challenges with integrated systems, though, is the extra testing burden they place on QA teams. A change in one area of an integrated system seems often to change things in other parts of the system. We need better regression testing tools, so that testing scripts can be maintained to automate the testing process in those other parts of the system where no changes are expected. One of the promises of the new "object oriented" technology is that integrated systems can be built using incorruptible, isolated modules, so as to minimize the "ripple" effect of system modifications.

Another method that has arisen to improve the systems development process is prototyping. Prototyping is usually done to show (rather than describe) how a user interface will function. But it should be done anytime you "smell a rat," that is, a technical problem where there are too many variables for a full analysis; in short, prototyping helps whenever you want to understand and validate your assumptions. It allows you to experiment cheaply, lower the risk, and plan for larger capacity, to compensate for the tendency of creeping scope to make your platform insufficient. To prototype properly, you should subdivide your application to the appropriate mix of, for example, number crunching, input/output, and video response.

We've also seen the parallel introduction of systems used more and more to help reduce the risk. Simply stated, don't turn off the old system until the new system has been proven in production. A parallel introduction will also help gain user acceptance. I know of one company, when it introduced E-mail, kept sending the users paper copies as well, since it knew that paper was "comfortable." Two months later, the users asked, "Why are you still sending us all this paper?"

PROJECT MANAGEMENT

In my experience as a systems project manager, one of the most difficult things I've found is the vast difference in productivity between the top programmer and the average one; I estimate it at 8:1. I would contend that this shocking difference is more extreme than you find in either clerical or actuarial staff. Systems people are technical staff, but in a different way than actuarial staff, although managing actuarial staff presents some of the same problems. Perhaps it is more appropriate for an actuarial vice president to manage systems people, than the more typical CFO type, who is more accustomed to managing clerical staff.

I have a cartoon about technical staff. The caption reads, "In a display of perverse brilliance, Carl the repairman mistakes a room humidifier for a mid-range computer but manages to tie it into the network anyway!"

How should a project manager schedule a project? Too often, the project manager becomes a slave to the schedule, charting progress in minute detail. I find that some of the software tools to keep track of the detailed schedule are so burdensome to maintain that I need a project secretary to keep it up-to-date; I prefer to spend my time managing the

expectations of the participants in face-to-face communication, a far more important task than maintaining the schedule.

As one project manager I know suggests, "Don't publish the schedule in too much detail, or people will start to believe it as gospel!" I have a cartoon showing a project manager in a suit, sitting on a park bench next to a "wino;" the project manager is expounding, "32 meetings, seven months of overtime, and 200 screens later, I finally said, 'Let's keep our fingers crossed.' But he didn't see the inherent humor in the situation."

Should the goals and milestones in the project schedule be soft and fuzzy, or hard and quantifiable? I believe that the overall expectation is the hardest to manage; try to leave the long-term goal as loose as possible. If it seems too long, my project management philosophy is to have people set their own dates under the assumption that everything goes right. Of course, problems do arise and the project falls behind. But motivation, I find, remains high until the scheduled date of the long-term goal becomes "laughable." At this stage I repeat the process of having them determine a new set of dates, since everyone wants to do a quality job, and will remain motivated as long as they are allowed to do so. The difficulty I've found with this process is not so much with the participants within the project who can see what I'm doing with this management style (it's pretty transparent, really), but with the external observers who understandably wonder if any project will come in on time. The secret is that the project executives must support this management style, and must quietly take a more realistic delivery time into their corporate planning cycle.

Short-term goals should be hard and quantifiable, by decomposing each phase into small manageable tasks, for which the participants set their own dates and know that they are held accountable.

SYSTEMS STAFFING AND STANDARDS

Should we replace the old mainframe systems staff, as we move to smaller client/server systems? IS professionals with traditional systems backgrounds can be very successful with the new concepts and technologies of computing. Their most important skill is that they bring the rigor and discipline required to implement business systems. Those who are successful will be the ones who see their value to the organization as taking new and existing business requirements into application systems, not those who are tied up in the detail of the minutiae, such as knowing every IBM error message ever created.

PCs are now a major mission-critical resource, and the IS department is typically the custodian of this corporate resource. So we've seen IS taking back control of the company's PCs.

PCs are now tied together in LANs, and we see the IS department's LAN administrator visit our PCs and help make them work together. But what does the LAN administrator do all day? He or she is doing the same work the IS department used to do on the mainframe (for example, training, maintenance, planning), but now the LAN administrator is more visible, since the work is being done among us, the users.

To leverage the company's investment in PCs and LAN technology, the IS department people are enforcing standards in hardware and software. They seem to be trying to dictate to us, the users, a personal work style, and we resent it. By standardizing though,

the company can significantly leverage its training and support resources, as well as negotiate from strength with hardware and software suppliers. PC technology is still immature; the number of PC support staff per 100 users is still far in excess of the number of mainframe support staff per 100 users. Yes, it is possible to make different operating systems and technologies talk together, but it is difficult and costly, and never as easy as they say it is. The CEO of a major oil company thought this principle was so important, he sent a private message on tens of thousands of videotapes to every employee in the company, saying, "Yes, imposing these standards may mean that you'll have to give up a feature that you really like, but you know, as leader of this company I have to say that the most important thing for our future success, is for everyone in this company to be able to work together as easily as possible."

IN-HOUSE DEVELOPMENT

When choosing to do in-house development, beware the lure of new technology. I know of one director of IS at an insurance company who says, "We must do something about our old system; it's falling apart!" Knowing that many successful companies are still using the same package, I asked, "What do you mean?" She said that the company had been growing so quickly that they were getting close to not being able to run the full batch cycle overnight, so they wanted to start a huge in-house development project to create an integrated client/server system to meet all of the company's administration needs. "But since you're not using anywhere near the largest mainframe computer, why can't you get a bigger mainframe and more disk space?" I asked. She responded, "But we haven't any more space in our computer room!" I asked myself, if the company has been growing so successfully, how can they embark on such a risky and costly in-house development project, but can't afford to expand the computer room? Before I could voice that, she offered what seemed to me to be the real reason: "Besides, I won't be able to keep my eager technical staff from leaving the company unless we have an interesting project using new technology!" Amazingly, this company has started their big development project.

Another risk I've seen, is how easy it is in a development project lasting several years, to always seem to be aiming at the next generation of future technology. When I was working at a software vendor, I once asked if the current design of a system in development would actually perform in production, when it seemed to be needlessly passing data back and forth without doing much with it. I was told that it was alright, because by the time the system was going to be in production in a company, larger processing power would be available and be the norm. Believe it or not, this conversation occurred six months before the first production implementations were going to be live. I realized that this same rationalization had been going on during the previous five years of development. No wonder that when this system finally did go into production, instead of six months later, it was literally six years later!

The lesson is to not just hope the system will perform. Successful developers know that the first implementations of a system never perform well; the users complain of slow response time and call the system a "pig." Successful implementations hide the "pig" by installing an oversized processor.

In-house development can get bogged down in "analysis paralysis," when a project team is able to start with a clean sheet of paper and dream about what could be possible. They keep analyzing and analyzing and seem never to come up with a firm foundation to build their system on. That's why I often recommend buying a software package, not as the

total solution out of the box, as oversold by the software salesmen and seemingly used as justification of the high cost, but rather as a working prototype from which the company's needs can be incorporated within a proven framework; even if the redesign is extensive, it's much easier to begin the first steps with a firm understanding of where you are.

David Kull in his excellent article, "Anatomy of a 4GL Disaster," (*The Interpreter,* IASA, February 11, 1986) notes that:

> Large projects almost always take on lives of their own. And participants, concentrating on their particular goals, lose sight of the big picture. Organizations can avoid this problem by providing for ongoing project review by a detached, objective observer. This QA must be provided by personnel with the technological expertise to recognize mis-steps, and the standing to enforce judgments that may run counter to the individual interests of team members.

METHODOLOGIES

I once talked to the chief architect of that system, which eventually remained in development for 11 years. I mentioned to him that usually when I approach a new system, I try to learn how it hangs together, for example, how the keys are built, and which are the important fields that drive the processing, or that direct the processing through an efficient path. But I told him, "I can't find them and none of the developers can tell me where they are!" Finally, in exasperation, I asked him "Where's the elegance in the design? Where's the art?" "Ah," he said, "That's where you're wrong! You're approaching the development in the wrong fashion. We're trying to engineer a system. You see we have a methodology for developing this system, and since you can't count on having brilliant people to design and develop a system, we have a methodology to lead systems people of average intelligence to successful completion!" I was stunned. I knew from personal experience and business case histories that small groups of brilliant people have achieved the most, and will continue to produce the effective and elegant systems we all want to use.

This chief architect was banking on the methodology working, which would lead to successful completion of this mission-critical system; he was betting the company on it.

I find that many people I talk to want to know what methodology I subscribe to. I'm at a loss for words. I usually try to say that I'm not beholden to any particular methodology; I can use whatever methodology they want. Unfortunately, this doesn't satisfy them.

A methodology is not a solution to problems arising in systems development projects. I still find that you still need great project leadership: Great project leadership can save you, even if you have a bad methodology, but a great methodology can't save you if you have bad leadership.

What do users want? Methodologies presume the users know what they want. I have a surprise for you. Users don't know what they want. They make it up! The systems analyst asks leading questions, after the user runs out of things to say. The user responds; these become requirements, which the user doesn't actually need. Specifications are developed through leading questions. No wonder we have so many problems satisfying users.

Frederick P. Brooks, Jr. wrote a book titled *Mythical-Man Month: Essays on Software Engineering* (Reading, MA: Addison-Wesley, 1975), which is still valid today. In just the same way as nine women can't make a baby in one month, no development team should have more than six participants or communication disintegrates. When a development project is in trouble, don't add more people. You should actually take people off, not only to create a more tightly-focused team, but also to get to the bottom of the difficulties and truly understand them.

DEVELOPMENT APPROACH

How then should you approach a development project? One well-respected consultant I know, says that he wants to staff a project with three programmer/analysts and two strong users. These users should be influence leaders, able to unilaterally change the business processes in their organizations. If he's told that they aren't available, then he refuses to accept the project, since he concludes the company doesn't want the project to succeed. The users are required to attend from 8:00 to 10:00 am everyday. Then he tells their manager to expect the two users to resign, and to accept their resignation, but that they can rescind their resignation by showing up the next morning at 8:00 a.m.

On Day 1, the programmer/analysts have been instructed not to lead the users, but to force the users to talk. They don't know the requirements, and neither does the IS staff. Those first two hours are allowed to drag out in near silence. The users run to their manager to complain that this is the most ridiculous project they ever worked on. From 10:00 a.m. to 5:00 p.m. the programmer/analysts create a screen prototype of what the users actually asked for. They've said so little, yet it's finished by 3:00 p.m. It's so ugly that the programming team leaves at 3:00 p.m. and doesn't come back.

On Day 2, the users show up with a "project suggestion list" about how the project should be run. The project manager thanks them and puts it away. Then the programming team shows the prototype, based on the clearly articulated user requirements. Not surprisingly, the users respond, "It's not what I want." The programming team replies, "Well, what do you want?"

Today's shrinking business cycles mean that the IS department doesn't have the time to interview the users and compile a list of user requirements. Users know what they want when they see it, and our job in IS is to facilitate the process, by giving immediate feedback of what they've articulated. We've shown them the screen prototype because, to a user, the system is the interface.

OUTSOURCING

If in-house development is so difficult, can we turn to consulting firms to provide assistance. Too often I've found consulting firms to be champions of exotic technology, whose sole reason for being seems to be to find a victim to try it out on. Instead of being "leading edge," they can leave you stranded on the "bleeding edge," and bathed in red (either red ink, red in the face, or maybe even with a pink slip!).

They can befuddle you with jargon. Indeed, some firms seem to be specialists at adopting each new set of buzzwords as they come into vogue, milking the novelty to their same old clients, and leaving them no better off, but ready for the next wave of jargon to come along. I have a cartoon, showing a professor admonishing his university students: "You

ask me the value of jargon? I'll tell you the value of jargon. To you, about ten grand a year!"

Assuming you can find a suitable firm to contract with, you still have the problem that the stakes are lower for "employees" on contract, than for internal staff. The short term of the contract may lead to short-term thinking.

Unfortunately, the ultimate cost of any system, either from the programming perspective or the user perspective, depends more on how easily it can be maintained than on how easy the initial implementation was. I know of one software vendor that follows all of its customers' requests. When the customer wants to keep costs down and to implement the system as quickly as possible, the vendor makes as few modifications as possible. To do this, the vendor splits each insurance plan the company sells into as many as 20, 40, or 60 separate plans on the computer system. By making a few, very inexpensive and minor modifications to the system, the vendor could cut this down dramatically. I would argue that these minor modifications should have been added to the base system years ago. But there's no way the customer could know how best to use the new system and when to exercise judgment; customer look to the software vendor. However, the software vendor will have finished the contract and no longer be on-site, by the time the customer realizes how difficult it is to maintain on the computer all of these separate plan records, and how this simple decision has percolated through the entire company, complicating the work of each and every user of the system.

One of the things that can be done in dealing with consulting firms is to structure their engagements carefully. David Kull observes:

> . . .the firm that writes the requirements and specifications should not actually implement the system. The federal government prohibits that arrangement in its computer projects. Splitting the two development phases builds checks into the process. The implementer can objectively judge the work of the specified. And because the specified knows it won't implement the system, it's not likely to build in expensive but unnecessary features.

Perhaps another way of controlling projects involving consulting firms is to insist upon fixed-price contracts. However, when problems arise on a locked-in contract, sometimes the financial pressures lead to declining workmanship. In spite of obvious pain being suffered, the company continues to hope for the best; unfortunately, usually the worst occurs, with the company having planned no provision for fullback or recovery, since it has its fixed-price contract to stand behind. David Kull offers a refreshing opinion:

> Large development projects frequently encounter legitimate, unexpected cost overruns. Rational compromises in such situations—perhaps a sharing of the extra expenses by developer and client for contract jobs or a budget revision for in-house projects—allow for happy endings to these situations.

SOFTWARE VENDORS

So if it's so difficult to do development, either in-house or with external consultants, why shouldn't we look to packages from software vendors? Well, these have their own set of problems. And I can talk from first-hand experience on both sides of the transaction.

You certainly will have heard about hardware and software, and some of you may have heard the term *firmware* (it's software that has been permanently written to a silicon chip,

that is, to hardware), but how many of you have heard of *vaporware*. Vaporware is software that doesn't exist yet; it's out there somewhere in the vapor. It could be called software under development. It may never work. When I joined that software vendor whose system was supposedly six months away from completion after five years of development (but was still six years away from completion), over 80 companies had bought the software, or rather the promise of the software. In retrospect, this was a triumph of marketing vaporware.

Many companies get caught up in the vendor's excitement and want to be the first to use the new software when it is completed. But, as David Kull writes:

> Two general principles apply to a pioneering software project. No software arrives in the marketplace bug-free and in final working form. The vendor provides fixes and adds functions and features—enhancements—when its developers complete them. This "maturing" process proceeds unpredictably. Prudent users, therefore, don't count on capabilities that are not in hand.

Realize that if you want to be first with new software from a vendor, you are in a joint venture, a partnership with the vendor to develop the software, but with much less control of the situation than if you did the development yourself.

VENDOR PROCESS

Generally, managers responsible for selection of packages are forced to make decisions in areas where they have no expertise. The nature of the beast, when using a software vendor, means that you go through the following steps:

1. Request for proposal (RFP), asking vendors to submit a proposal if their system meets your needs, as specified in the RFP.
2. Response from the vendor, almost always favorably. As Donald Belfall says in the article "Guilty until Proven Innocent" (*Software Report*, February 20, 1986), "Any software salesman can improve the appearance of his product line through a series of unqualified responses, half-truths, loaded reference contact lists, and tailored demonstrations." Is that ever true! In my experience with that software vendor I couldn't believe how we would answer half the question, or answer a different question than the one asked, all so we could say "Yes, our system does it!" And when I questioned the ethics of this, I was told that all the other companies responding would be doing the same thing, and since the client would be better off with our software, it was alright. Software salesmen basically tell you whatever they think you want to hear.
3. Evaluate the response from the vendor, and investigate, by checking references and seeing a demonstration. Too many times demonstrations are "canned." Donald Belfall again: "The well-prepared checklist of an organizations' requirements on a feature basis will force qualifications to positive responses when necessary. Every assertion made by a salesman should be obtained in writing." We'll look at the evaluation process in more detail later.
4. Negotiate and sign the contract. The vendor is amazingly accommodating until you sign the contract; after that, he thinks you're hooked (that is, committed), and he can relax, since the marketing phase is finished and you are comfortable with the inflated expectations he's left you with. Regarding the contract, Donald Belfall writes: "All vendors should be informed at the beginning of the selection exercise that any assertions made will be appended to the license agreement. This will include claims regarding software functionality and the guarantee that

reference lists submitted were complete." And as stated in the article "Staying out of the Black Hole," by Bruce Brickman (*Best's Review*, February 1986):

> Instead of bargaining, technical personnel often buy on the vendor's terms. . . the vendor's contract becomes the binding document. Naturally, in some cases, the contract can reflect the vendor's objectives of no link between performance and payment, no standards for training, inadequate maintenance and no recourse for performance failure.

We as insurance professionals don't understand the vendor's business and his practices. For the most part, our salaries are paid by the renewal premiums rolling in almost automatically from our policyholders. Software vendors lead a hand-to-mouth existence, scratching for every dollar to stay alive. They make it sound as if the license fee is not negotiable, that it represents their valuable intellectual property. Well, I have news for you: vendors consider the license fee "gravy" that goes immediately to profit (less the salesman's 30% commission), and is priced at whatever they think the "traffic will bear." Software vendors make their money on the services they offer: implementation, modification, enhancement, training and maintenance fees. They have an overall revenue target for you to pay. So if you start to negotiate for fewer services from them, the license fee will suddenly become very firm; and of course, the reverse is true, too. My best advice to you, is to get outside help in negotiating and signing the contract.

5. Implementation study and project plan. Now that the marketing phase is finished, the client is passed over to the consulting arm of the software vendor. I was one of these people. I hate to say it, but my real job was to lower the inflated expectations the sales process left the client with, and to do it in small steps, so that the client wasn't disappointed all at once, and didn't lose faith in the company. Dragging out the implementation, with many phases, made my job easier.

6. Implementation, modification and enhancement. I've already discussed how, if you don't have the right priorities in this phase, you may end up with an unmaintainable system. It's difficult to have the best of the vendor's staff assigned to your project, unless you have some sort of "clout" with the vendor. And the vendor doesn't consider the terms of the contract your "clout."

How would I change the process? I feel the vendor should leave the system and its documentation on your site for your review. You can learn all you want to, such as how well it performs under stress, how it's internally structured, and where you think the modification dollars should be spent.

VENDOR EVALUATION

Let's talk more about evaluating the vendor. This is probably the most difficult step. In hindsight, where do you think the most mistakes are made in evaluating the vendor, that is, where does the actual experience, in hindsight, vary the most from what was expected? Surprisingly, it's in the following order, from most off the mark, to least:

1. Quality
2. Risk
3. Time
4. Cost

VENDOR QUALITY

I'd like to quote extensively from Richard Smith, Jr.'s excellent article "Evaluation and Selection of Application Software Packages" (*The Interpreter*, IASA, June, 1978). It's a very candid article, especially since Mr. Smith was working for a software vendor when he wrote the article:

> A key contributing factor to "Quality" being ranked as most frequently miscalculated is a lack of focus on the basic purpose of a good software package. A software package typically does not represent a total solution to a problem but is a key component to be utilized by a company to derive the total solution. The large application package generally must be "adapted" and often "extended" to a varying degree by data processing and user personnel. The package must coalesce with a company framework of manual work flows.
>
> The problem is not that the great majority of decision-making teams do not recognize, in general terms, the role of the software package. The problem often is a lack of focus on or lack of understanding of what constitutes good generalized software in specific, measurable terms. This problem often surfaces as a preoccupation with the existence, or lack thereof, of detailed functional capability, accompanied by insufficient attention being given to the overall structure and capability of the system and the ease with which it can be made to respond so that it provides all required functions.
>
> An effective decision-making team must have an in-depth grasp of what constitutes good generalized application software. Unfortunately, this is often not the case. As an example, a decision team is generally sent forth to do an evaluation with the charge to "make sure the package is flexible." The concept of "flexibility" has almost become a cliche within the software industry. The reader can rest assured that every application software package on the market will be advertised as "flexible." In reality, there are wide differences in the "flexibility" of application software packages. In addition, the very concept of "flexibility" is often not even defined sufficiently to allow an evaluation to be made. . .
>
> All application packages of any significant size require some degree of "tailoring" in order to adapt to a specific operational environment. Such tailoring can be as simple as entering a few numerical values into edit tables to as complicated as requiring significant assembly language reprogramming of existing program code. The ease with which a program can be tailored to fit a specific user environment and "how" this tailoring is accomplished are two of the most significant factors separating good software packages from poor ones.
>
> Adaptability should be built into a system from the beginning through anticipating the vast majority of changes that will be required when going from one user environment to another. . .
>
> Too often an evaluation team will not delve deeply enough into this aspect of the system. The question is not the simple existence of some table driven routines, because every large application package will have table driven routines that can be paraded past the evaluation team, but the crucial issue is the extent of generalized concepts and the degree to which the unique aspects of the company can be

accommodated without reprogramming efforts. Or, I might add, accommodated with relatively simple programming efforts.

VENDOR RISK

Richard Smith, Jr., asks some pointed questions regarding the risk component:

> How long has the vendor been in the software package business? All software vendors go through similar learning curves and typically make the same mistakes in their early years such as:
>
> 1. overcommitting their resources through outselling their ability to install, or promising system modifications within timeframes that they cannot meet.
> 2. not fully appreciating the system design concepts that must be utilized to create high quality software packages.
> 3. inadequate appreciation of the need to evolve packages through a series of "digestible" system releases over the life of the product.
> 4. lack of appreciation of the highly professional organization that must be in place to service a large base of clients.
>
> What is the history of the package? How many times has the package been successfully installed? How large is the vendor in terms of revenue, personnel and financial backing?

When I was working for that software vendor, I didn't understand why the vendors go to such great lengths to explain their financial situation. We insurance professionals don't pay enough attention to this, since we don't understand the software vendor business, as I discussed before. Software vendors go where the money is, and if you run out of money, they go somewhere else. We think we'll be protected by the contract, but software vendors can afford to have one company mad at them; they just give you your money back, leaving you with wasted time and effort on your part. But they can't afford to have all of their clients mad at them at the same time.

TESTING

Bruce Brickman offers some further advice about the testing phase of implementing a package:

> Testing a product after payment can mean fighting with an unresponsive vendor. The key is to link payment to testing, thereby requiring an investment by the vendor in assuring performance. This means withholding payment until performance is proven.

Certainly, mature systems have fewer bugs. In fact, instead of using the first release of the most recent version of a vendor's software, I've been known to have our company choose instead to install the final release of the previous version of the vendor's software. In hindsight, everyone from the vendor, other user companies, to our own staff, exclaimed how wise we were to do so.

And don't forget that with today's integrated systems, lots of regression testing is required to make sure that modifications and enhancements in one part of the system don't adversely affect another part.

CONCLUSION

Our difficulties with technology are nothing new. Often, implementing new technologies have brought other problems and effects, other than what was intended.

I'd like to close with a quotation from Neil Postman's book, *Technopoly*, which is subtitled, *The Surrender of Culture to Technology* (New York: Random House, Inc., 1993):

> The invention of the mechanical clock . . . had its origin in the Benedictine monasteries of the twelfth and thirteenth centuries. The impetus behind the invention was to provide a more or less precise regularity to the routines of the monasteries, which required, among other things, seven periods of devotion during the course of the day. The bells of the monastery were to be rung to signal the canonical hours; the mechanical clock was the technology that could provide precision to these rituals of devotion. And indeed it did. But what the monks did not foresee was that the clock is a means not merely of keeping track of the hours but also of synchronizing and controlling the actions of men. And thus, by the middle of the fourteenth century, the clock had moved outside the walls of the monastery, and brought a new and precise regularity to the life of the workman and the merchant. "The mechanical clock," as Lewis Mumford wrote, "made possible the idea of regular production, regular working hours and a standardized product." In short, without the clock, capitalism would have been quite impossible. The paradox, the surprise, and the wonder are that the clock was invented by men who wanted to devote themselves more rigorously to God; it ended as the technology of greatest use to men who wished to devote themselves to the accumulation of money.

May all your technological projects turn out the way you intend!