RECORD, Volume 23, No. 3*

Washington Annual Meeting
October 26–29, 1997

Session 27PD
Radical Product Implementation

Track: Science/Product Development
Key words: Computer Systems

Moderator: DAVID C. MILLER
Panelists: ROBERT G. FAIRCHILD†
JEFFREY M. ROBINSON‡
Recorder: DAVID C. MILLER

Summary: The current competitive and regulatory environments make it difficult to quickly deliver desirable products to the marketplace. How can the system development aspects of products and service implementation be streamlined in order to meet these requirements?

Mr. David C. Miller: Bob Fairchild is a Senior System Manager with the Prudential. He's been with Prudential for 24 years. He has spent the last 12 years in information systems as a developer and project leader. Bob has been the project manager of a client-server development project for the past two years.

Jeffrey M. Robinson is president of Life Insurance Financial Essentials/HAS, which has for more than 20 years been providing consulting actuarial management and systems implementation services to life insurance companies, other consulting actuaries, and data processing software firms. He's been an actuarial consultant for more than 28 years, and has served as a company officer for over 7 years. He has worked with more than 51 life insurance companies at 6 fraternals located in 7 different countries. He was chairman of the data group and the new schedule development group of the industry committee working on the modernization of New York's Section 4228. He has been providing and defending the small companies' perspective to that committee.

*Copyright ©1998, Society of Actuaries

†Mr. Fairchild, not a member of the sponsoring organizations, is Senior System Manager, IIG OPS&SY of the Prudential Ins. Co. of America in Newark, NJ.

‡Mr. Robinson, not a member of the sponsoring organizations, is President of Life Insurance Financial Essentials HAS in Parsippany, NJ.

I'm Dave Miller. I'm a consulting actuary with Actuarial Science Associates, Inc., and I'll serve as the moderator and as a panelist. I specialize in life insurance and annuity product development, and product and systems implementation.

I'd like to begin our discussion with a quote from *The National Underwriter*, February 17, 1997. I quote, "In the insurance industry change is as constant as breathing, and the demand to create new products faster increases nearly every time you inhale."

Insurance products today have extremely short life cycles. Many companies are constantly reengineering their product development processes in order to get their products to market more quickly. I believe there are at least two environments that contribute or drive the need for frequent and timely product delivery. One is the competitive environment. Companies are with great frequency designing and redesigning products that are customized to meet a consumer's needs within a certain niche. A new bell or whistle or product feature may be needed simply to keep pace with the competition, much less get ahead of it.

The second environment is the regulatory environment, and I'll throw tax in there as well. For example, the new illustration regulation creates the need for product and system revisions. Regulations dramatically impact companies' business plans and their approach to the marketplace. One of the major critical path items in product implementations is the need to build, enhance, and modify your systems. The ideal is to be nimble and to react to marketplace and regulatory change.

We'd like to discuss an approach, which has been around for some time, but is just recently being used more by insurance companies to address these issues. This approach is called Rapid Application Development (RAD). Our first panelist, Bob Fairchild, is going to lay the groundwork by explaining what RAD is.

Mr. Robert G. Fairchild: I'd like to discuss the differences between, or at least compare RAD with more traditional approaches. I've been in information systems for about 12 years, but only for the last 2 have I been doing anything other than mainframe system work. I was recently introduced to RAD too. I'd like to take a few minutes and talk about traditional development and compare that with RAD. Then I'd like to share some experiences that I've had with our RAD project.

By traditional development I'm referring to the formal and structured format used to develop systems. There are several tasks or phases involved in systems development; in a traditional format each one must be completed before the next one begins. Why should we search for an alternative? Is there something wrong with traditional development? Not in most instances. But there are some situations where we would look to see if we could get the product to the market more rapidly, as Dave had mentioned earlier.

The traditional approach certainly takes a lot of time from the start of the project to when the actual product is implemented. And sometimes this can cause problems. Mostly we have situations where the user requirements are no longer valid by the time the system is finally completed. But what I consider even more important than that, or at least equally important, is that the end user does not share in the development process. The user appears early in the process and provides the detailed business requirements, but then disappears until the very end. Many times it isn't until the system is implemented that the end user sees it for the first time.

So what is RAD? It's a development life cycle that is designed to give faster development and higher quality results. RAD is not a cure-all. But it definitely is a worthwhile process for specific situations. If it's done right we can deliver something much faster than what we have been used to.

I'd like to take a couple of minutes to talk about these specific items and to compare the traditional and RAD approaches.

The traditional approach is much more formal; RAD is more informal. With traditional we expect the end users to prepare well-defined and detailed requirement documents, and in many cases the end result can be an inch-thick paper. We expect the system developers to take those detailed requirements and produce a well-defined and detailed design document for the entire system. Unfortunately these are done independently, and there's very little involvement by the system developers when the detailed requirement documents are produced. And there's very little or no involvement from the end users when the system design is being put together.

RAD approaches this in a slightly more informal way. Part of the RAD process is to have a Joint Application Design (JAD) workshop. This is where the project scope is defined and both end users and developers are involved in the process. It can also include prototyping and modeling. A RAD session usually takes about a week. The end result, ideally, is a project scope that is defined at a high level; the details will follow later.

Again, the traditional approach is usually very structured. RAD is much more iterative. With the traditional approach the detailed documents that are developed early in the process serve as a foundation for staged development. Each stage must be completed before the next stage begins. In the RAD process much of the work is done in parallel. We will develop data models and process models, and also prototype and develop some front-end screens in parallel. Much of this starts in the JAD sessions.

One of the key things about a RAD project is that you do have incremental delivery. The end user does not need to wait until the end of the entire system development project to see a finished product. The RAD process is set up so that functional objects are implemented throughout the process. Development testing and implementation happen concurrently. And with the direct user involvement there is constant communication. As systems are developed they are refined based on user input and feedback. This is all happens during the development process.

We found on our project that the results were so much more beneficial and meaningful because as we were developing the product the end users were right there watching us, giving us feedback, and telling us what they liked and didn't like, and we were able to respond to that immediately. With the traditional process, the end users' involvement is simply in the beginning, and they don't see us again until 18 months later or whatever the time frame is.

This is important too, because many times, in a more traditional approach, target dates are generally pushed back in order to allow for the completion of the entire system. From a RAD perspective the focus is more on functionality, and there are few times when a target date will be pushed back in a RAD project rather than removing functionality in order to meet the target date.

These are common characteristics of RAD. There is rapid implementation of the system. The user participation is key, and it occurs throughout the entire life cycle. This enhances the communication between the end users and the developers. The developers have much more of an idea of exactly what it is the users are looking for, and the users are not surprised about the end product. RAD usually needs, and in an ideal situation it calls for, powerful development tools. In our situation we probably did not use development tools that most people would consider completely RAD. We had other factors that addressed that issue.

Having reusable components is important. Obviously, the more you can reuse and the more common modules you can develop, the less you need to continue developing, because you have something there that may need just some minor modifications.

Timebox development is important in RAD. This is where we define specific functionality that will be developed in a fixed and short amount of time. This requires a lot of commitment between end users and the developers to succeed.

With the full participation of the end users, we have direct, frequent, and ongoing communication. The end users are there. They are there throughout the entire process. They don't disappear. As we prototyped throughout the process they were there to give us feedback. It's very critical.

There are tools, such as Power Builder and Delphi, that are designed for RAD development. These tools and others like them tend to allow the developer to develop a complete database with minimal coding. There's a lot of background coding done by the tools, which are useful in following through in this type of process. Again, reusable components are important. Try to design modules so that you can reuse them in other areas of the system. Reporting is important. Although many reporting modules can be used for many reports, make sure the one you select is flexible enough.

Timebox development is critical in RAD because the focus is on developing and implementing functionality. One of the critical items here, however, is that there must be a stable project plan. Scope can kill a RAD project. This is where it takes a solid commitment by the end users to make sure that they don't allow scope to enter in here.

Two years ago I was introduced to the RAD process. I'll just talk a couple of minutes about my experiences. My internal client was looking for a system to replace its legacy mainframe system. Ideally it wanted a client-server system to replace both its administration and its illustration systems. It's a COLI product. The following is a brief description of the parameters:

- 20 users
- 29 group contracts; 115,000 lives
- $3 billion assets under management

What were our reasons for RAD? The first reason was to prove that it could be done. We had an executive who was very much behind the RAD process. It is critical that you have upper management support for this because it is nontraditional. It’s sometimes very difficult for upper management to accept the RAD process. We were fortunate enough to have an executive who was driving the process.

We wanted to bring the product to market quickly. We've had many systems, that by the time they were implemented some of the business rules had changed making part of those systems no longer applicable.

We were fortunate to have knowledgeable end users who were assigned to the project on a full-time basis. Again, it's critical. It's also not very easy to do. But we were in a good position right at the start of this project, and we had those end users available to us.

We had a very experienced RAD development team. This team that we chose to develop the system had been experienced in the RAD process for several years. They had just finished a RAD project elsewhere. The team members knew each other's strengths and weaknesses.

Earlier I had said that we used some tools that may not necessarily be considered RAD tools. We used SQL as our stored procedural language. We used Visual Basic for our front end. Visual Basic can be considered a RAD tool. SQL usually is not. But the fact that we had this experienced RAD team more than made up for the fact that we were not using a tool such as Power Builder. This was the makeup of our team when we started.

As for our results, we delivered a functional system in five months. Prior to starting this development project, we had received estimates of as high as 18 months to develop the same system. We also got out of it a highly motivated team and very happy end users. They didn't have to wait 18 months to get their system. It also laid the foundation for future phases, some of which are still RAD, and others that have gone to a more traditional approach.

We're continuing to use RAD for our illustration development. Prudential is currently involved in other RAD projects, and I believe it's going to continue for specific projects. It definitely has been proved and it is backed by an executive. As long as she's behind us I think that this approach is going to stay in Prudential.

Mr. Miller: So far we have defined RAD, or at least a school book version of RAD. And we've heard a little bit about how Prudential is utilizing RAD. Jeff and I are working with Bob on Prudential's project. We were wondering who else is using RAD? Prudential realized some real success here and wanted to learn if other companies were using it. We did some research and got a list of companies that were using RAD. I spoke to four of the companies and I want to share with you some of their experiences. They gave me permission to do so.

I focused mainly on three questions. What did you use RAD to build? Why did you choose a RAD approach? What were the results? After I go through the answer to these questions, I'd like to explore the critical factors for RAD success and also the advantages and disadvantages of RAD.

The four companies are General Accident, Great-West Life and Annuity, Northwestern Mutual, and Relia Star Financial Corp. These are four very different companies, but they have all used RAD in some way. Wayne Ratz was the leader of the General Accidents project, and I spoke with him. He has a quote, which I think captures the essential difference between RAD and traditional (TRAD). Let me read this quote to you, "Developing code with RAD is like building with Lego blocks as opposed to concrete and steel. If you build with concrete and steel you labor for months about the architecture and the structural formation. With Lego they're reasonably steady, but if you don't like what you built you pull it apart and rebuild it."

General Accident used RAD to build a 3,100 function point processing application for its business owners' policy called Business Elite. It chose RAD because it was faced with all the old, inflexible systems and enormous cost and time estimates to develop new systems. And, in general, it was not meeting the expectations of its users. So it set out to create a new development environment. That's why it went with RAD. Its goals were to reduce delivery time and cost, to improve quality, and to improve the flexibility of the system to allow for more rapid modifications going forward.

In 1994 General Accident used a RAD team of 15 to 20 people. It constructed timeboxes in six- to eight-week intervals. It rolled the system out in ten months, about four times faster than TRAD. This is a real-time system. Wayne tells me that it built this system on two previous occasions on a batch basis using a TRAD approach. He said both efforts took more than three years, and both used more resources than this RAD effort. General Accident undertook its third RAD effort in the fall of 1995. This was a commercial auto processing system that went live during the spring of 1997, so that effort took 18 months to complete.

Great-West Life and Annuity used RAD to build a new billing system for its group life and health business. It noticed that the TRAD effort for its eligibility system projects was struggling. The users were trying to get a look and feel from screened designs on paper. Great-West opted instead for RAD on the billing system project so that it could furnish the users with working prototypes. In general, users were not involved in the process. Great-West thought RAD would be a way to get more user involvement.

What were Great-West's results? Phase 1 included 200 screens and 50 batch processes. These were completed in two years. Matt Kramer, who was the leader on that project, estimated that it probably would have taken three years with the TRAD approach. It also achieved more user involvement. Matt says that involvement is more second nature now and users are more integrated in the process.

Phase 2 had a similar scope and it took only a year. Since then, Great-West's third version of RAD has evolved into a hybrid RAD and TRAD. They've seen situations where there is a need for detailed specifications.

I spoke with Steve Strommen at Northwestern Mutual, which used RAD to build its corporate financial modeling system. It's written in C++, which is interesting because C++ is not considered a RAD tool. It shows you can still follow this approach using non-RAD tools. It has developed and enhanced this model over the past ten years.

I asked Steve, "why did you choose to go RAD in this situation?" He said, Northwestern Mutual didn't call it RAD back then, but in retrospect it was RAD. It was faced with very limited resources and a very short timeframe and RAD was the only way it was going to get it done. The result was the RAD team, which consisted of 1 FSA and 2 systems developers with 10 years of business experience, delivered the first version in under 12 months. Recently it developed a projection module for a new line of business, variable life, in a three-month period. Steve estimates that this RAD approach achieves results at least twice as fast as the TRAD approach.

Reliastar Financial Corp. used RAD to develop an agents' workstation called the Reliastar Connection. This system is an integrated set of products for the sales force that allows the agents to receive updates on new business in underwriting, and provides the sales illustrations, reports and e-mail in a uniform manner.

There are basically three reasons Reliastar wanted to develop the connection with RAD. The first is it wanted to avoid being painted in a corner. Since business problems are continually changing and the technology landscape is changing rapidly as well, it did not want to get committed into a total solution and not be in a position to take advantage of new things that came along.

Also, Reliastar had a vision to roll out new releases about once a quarter. It wanted to start small with this application and build, which is very conducive to RAD. Also, it wanted to engage its distribution on a regular basis. RAD provided a reason to routinely sit down with its distribution and get feedback to ask them, "What's going on out there? What are your needs? What do you want this system to do in engaging conversations?" I thought this was very creative.

Reliastar began this project in 1995 and it got 4 releases out in 12 months. This is about four times faster than it would have taken using a TRAD approach. The RAD team consisted of three to five systems people and two to three end users. They have continued meeting with the field force and enhancing this workstation on a quarterly basis.

You've heard stories now of five different insurers that have used RAD for a variety of different applications. You might be thinking, how can I be sure that RAD would work for me in my company and on our projects?

What I'd like to talk about now are some of the critical factors needed for RAD to succeed. These are based upon our observations after speaking with some of these companies. The critical factors are company culture, user commitment, a multitalented team, and timebox development.

The biggest challenge for most insurance companies will be a culture that supports RAD because RAD is typically countercultural. A big cultural issue is that initially you get less functionality which, in turn, does not solve the entire problem at once. It's the acceptance of less functionality. Say we get 20% of the required functionality initially, with the 20% growing over time. Most likely it will be difficult to get your management comfortable with this approach to overcome the general feeling that RAD will let things fall through the cracks. It will be a challenge to get your management to be comfortable with the seemingly informal process, and to accept that RAD usually involves only a few people who are experts versus assembling a large project team.

RAD crosses organizational boundaries that may support very different policies, standards, and philosophies. All disciplines must buy into RAD, because all disciplines are involved in RAD. For example, will your information technologists work without detailed and final specifications? You will probably face some opposition. There will be people in your organization who will be open-minded, but there are always some people who can't let go of the processes they've held allegiance to for years.

As Bob mentioned, it's critical to have the support of someone who's in a senior management position. You need this person to understand the process, to help support the change, and to facilitate the change. Many of the companies that I've spoken with recounted how RAD efforts were weakened or abandoned when the "evangelist" in senior management left the scene.

Finally, it's better if you have minimal bureaucratic structure or if there is a way to make the procedures minimal or nonexistent. Your RAD team needs to be empowered without the need for frequent recourse to higher-level management. Management should be careful not to impede the process. Its role is to empower, to make decisions, and to resolve conflicts.

The second critical element is user commitment, that users are actively involved and available. This is something that we'll stress for the rest of the day. Early and frequent input by the end-user is the foundation of the RAD process.

You need to have fully dedicated knowledgeable resources, which might be a challenge. It can't be the kind of project where you get someone from the business area who doesn't really know the business or what you need from the system, and who can't make decisions on his or her own. You really need a seasoned person. But the argument you hear is, "Hey, I got a business to run. I can't give you my best person." I understand that. I think one way around that is the business area should have two or three of these experts who can share this responsibility and still tend to the day-to-day running of the business. But it's essential that you have an expert.

You need the users to define and agree on the scope of the project. This is a blocking and tackling exercise no matter what approach is taken. Lack of scope definition and "scope creep" are, as Bob mentioned, typical contributors to a project that spins out of control and even worse, a project that loses credibility with management. The users have to be able to make decisions quickly. When an unanticipated crossroad is reached in your project, decision lags must be avoided.

The third critical element is having the right people. You need a multitalented team with various disciplines represented. Most of the companies we've talked with had the following team layout: project manager, team leaders, business analysts, users, and a quality assurance team. All the members of the team need to have certain traits. They need to be experts and to know their subject matter, whether it be technology or the business. They need to be motivated, fully dedicated, and empowered.

Also, smaller, highly-integrated teams tend to be more successful with RAD. I believe this is because communication is so important in RAD. As you add more people the division of labor gets quite thin, and the communication dynamics start breaking down.

The last element is timebox development. John Mittlertadt at Reliastar believes this is the essence of RAD. He defines a timebox as, "What can I do in 90 days?" Timeboxing works by focusing on when a business objective will be met as opposed to the tasks that contribute to its delivery. It forces time to be nonnegotiable in your intermediate deliverables. Timeboxing combats endless iteration, which occurs as users see the prototype and want to make changes. You could be making changes for days or weeks. Instead, make the changes during the next timebox so that you can keep the project under control.

Timeboxing also mandates frequent review and feedback. It defines a date when portions of the completed deliverable are due for team review. Feedback from the team can then be incorporated in the next timebox. Timeboxing contributes to a better quality product. It's fairly certain that as the volume of code in the software gets larger, the amount of software errors will increase almost exponentially. Timeboxing is a way to work with subsets and smaller pieces of code that you test after each deliverable. It allows even larger applications to be developed and tested in small chunks throughout the development process. In addition, any problems that are found are much easier to debug as compared to the TRAD approach, where testing occurs at the very end of the completed system when there's much more untested code to analyze.

I believe if you have all these elements in place—the culture, user commitment, the right team, and timebox development—then I think your chances of succeeding with RAD are quite good.

Let's summarize some of the advantages of RAD. First, user involvement is very important. RAD allows the users to take ownership. Instead of being critics, users take ownership and pride in the success of the development effort. In turn, their involvement drives the process.

Second, there's shorter development time. RAD gets you into development earlier in the process. With the traditional approach there is paralysis of analysis. You try to get every detail perfect before you even begin. With RAD, rather then spending a great deal of time up front locking in designs, you can begin development almost immediately and revise the designs as you see them. Because you get out of the initial development stage earlier, and you're using smaller self-contained pieces, you start working with the system and get feedback, which will be incorporated in the next phase.

Getting feedback occurs throughout the entire process, which helps contribute to better project control. Let me share a scenario with you, which you may or may not have experienced. This may be somewhat exaggerated. With a large system project you spend a great deal of time writing specifications that you have to make perfect. Then the programmers take those specifications and build the system. Some amount of time lapses—it could be a couple of years—and the programmers return with something less than what the users wanted. Why? Because the specifications weren't perfect, or at least the programmers' interpretation of them weren't perfect. The programmers then go on to another previously scheduled project, leaving it up to the user to fix a flawed system. Management has now spent a significant amount of time, resources, and money on a system that cannot be used.

With RAD, review begins much earlier in the process. The users provide feedback after the first timebox, so you have checkpoints along the way, which keep the project on track. You have those checkpoints before a lot of money has been invested, which leads to better project control.

RAD provides systems that are more adaptable to an evolving business. In the current environment it's very difficult to lock in a system design two years before you're going to use the system. With RAD each checkpoint offers an opportunity to keep the system relevant.

Another advantage of RAD is that it produces a higher quality product because of the timeboxing process. You're more likely to give the users something that they like because they're involved and they're monitoring the whole process. And finally, with RAD there is a motivational advantage and your experts are empowered to succeed.

Let's look at the disadvantages of RAD. Prototyping is a double-edged sword, because it can lead to endless iteration, as I explained before. This is why timeboxing is so critical, and why agreement with users to lock in the scope is so critical. Team burnout can happen because the pace of RAD projects tends to be very intense with short deadlines. This is especially true if the project extends over a long period of time. Burnout can be managed by setting reasonable timeboxes and being willing to move functionality, if needed.

Your documentation may suffer in RAD. Documentation is usually the first thing that gets sacrificed because the emphasis in RAD is to get started now and write it down later. There's often a steep learning curve, both in terms of educating your company and using some of the RAD tools. Many companies such as Prudential have hired consultants who know the RAD tools, rather than waiting for their employees to get up-to-speed, which can take two or three years, and potentially missing the marketplace. It's common for consultants who are experts with these RAD tools to work side by side with the employees and then leave when the employees are up to speed.

There's possibly a large up-front investment needed to purchase some of the RAD tools. RAD is not conducive to high-turnover situations. Because RAD requires a few experts, it may take a large investment to replace that knowledge if one of the experts leaves the project. The TRAD approach tends to lend itself better to documentation and detail specifications because it's easier for someone new to pick up where the last person left off.

Some of the feedback we received from RAD advocates included fewer resources, faster overall delivery, and lower cost. However, people who have used RAD but aren't passionate about it would argue that the experts need to be around longer. In a TRAD approach, you start with a larger team, but that team gets weaned as the system goes into maintenance mode.

RAD delivers the ultimate system sooner. The con argument is that the iteration and constant review process may cause the project to take more time than the TRAD approach, or at least the same amount of time. Is it lower cost? Well, the premise of school-book RAD is to deliver a system in less time with less people, which equates to a lower cost. It makes a lot of sense; however, there are some studies that show that RAD is cheaper as long as your project stays within a short-time frame—say, under a year. As the project gets longer, it becomes more expensive than TRAD usually because the experts cost more money.

A lot depends on your individual situation. Which brings us to the question: Is RAD the right approach for your implementation project?

Mr. Jeffrey M. Robinson: I'm going to discuss how you go about doing your project. My remarks are based on my 35 years of writing specifications for applications for many different companies, and in many different countries.

Actuaries think they're technicians; data-processing people think they're technicians. Can you sense a little tension here? In the past, actuaries defaulted their role in these projects and left because they weren't getting what they wanted. But if they left what happens? I come in. I mean I love it. But the actuary is the best person to work on one of these projects. Who knows best about what happens in an insurance company in an overall sense? The actuary. If you leave you're going to get something less than what you wanted. So stay in and fight.

To stand up to the data processing department you really have to fight. Hopefully you can work with the department, but it's not always easy. Stand up for what you want because otherwise, you won't get it.

We know what RAD is, and we know the RAD experiences of five insurers. We know the RAD success factors, and we know the advantages and disadvantages of RAD. But now it's time to choose. What do we do? Well, we take those critical factors and we break them down.

The first is the project scope. If it's a big, integrated project, I think chances of getting RAD are going to be hard. Don't even think about it. If it's big but there are little pieces that you can break off—if it's modular—then you have a chance. If it's small and independent, it's probably a good environment for RAD.

If your project is small or mostly-modular you have a very good chance. If it's stand-alone, where you don't have to integrate with many other systems, it's got a good chance. But you can mix and match, as we did on the Prudential job. Some parts you can do RAD, some parts you can do TRAD. But you should look at the particular project and think about whether RAD will be acceptable.

Culture is a very important thing. You have to figure out who's running the job. Who's the controlling influence? If it's TRAD, you know those people are going to bog you down with specifications and requirements, and before you know it, it's two years later and you're still writing specifications. We had that particular experience. So the chances of RAD in a traditional life insurance information services (IS) department are slim. You won't ever get it through. In the actuarial department it depends on how stiff they are and where they're coming from. You may have a 50% chance. If it's a new company, they don't know much. So it's easier to use a nontraditional approach. We worked with a six-or seven-year-old department that had no tradition or history. It was an excellent choice for RAD.

And if it's the agency department—well how many of you have tried to get specifications from the marketing or agency department? It's just impossible. You just can't do it. Unless you give them the specifications to choose from, they'll complain and you won't get anything. So an agency-type situation has a very, very good chance of using RAD. That's where it makes a lot of sense.

If your product is a Par Whole Life, there are very few new things around with regard to it. I think maybe Dave won't agree with me, and it depends upon your particular job and other criteria, but I wouldn't say that's the best product for RAD—except maybe in illustration systems. If the product was new, innovative, and had no tradition, such as trust owned life insurance which is the product we were working with. There were maybe 28 people in the department who managed $3 billion of assets, and had no tradition. This was an excellent choice for RAD.

Regarding the platform, what do you associate traditional IS departments with? The mainframe, because these departments can use an approach they like, the paperwork that goes with it, and the control. It's not a place where you start with RAD. With a mini it depends; you have an AS/400 maybe you know that those people are not steeped in tradition as the mainframe people are. A client server environment is much more conducive to RAD. Client server is informal; that's why people like it. The Prudential project had a client-server environment. You can do things more quickly, with a client-server platform, which is a necessity with RAD.

Management is very important. It never hurts to have a strong godfather. Actually, we had a fairy godmother on our project. She was very strong. Interestingly, she didn't come from an insurance company; she came from a bank. And the team that she was using came from that bank. She said, "this will be done," and it was completed in five months.

My background is with small companies. Small companies usually have a dictator. If they want something they get it. It's good to have a dictator in that environment. However, the people working on a RAD project have to be empowered. They have to be able to make decisions as they go. It's not a type of approach where you have to go through six layers of management before you get an answer. You're doing this approach essentially in an ad hoc environment. One of the strengths of the RAD approach is that you motivate your users, by empowering them

Concerning morale, in most of the data processing projects I've been on, the morale goes up and down like a yo-yo. Once you start losing morale and the people don't think the project is going to get done, you lose them. You've got to look forward. You've got to keep moving. You've got to be flexible. A motto that I like to use is, it's not what we did wrong, but how well we recover that counts. You're always iterating in RAD, if it doesn't work you try something new. In a TRAD approach, as Dave said, you go a year and nobody sees anything and then the system comes out and if it's lousy, what do you do then? Well, it's very, very difficult. With RAD you can take small steps. You put a screen up, you show it to the users, and the users tell you what to do. It they like it, fine. If they don't, you do something different.

What are your objectives? If you want to get started quickly and show progress early, RAD is the approach. TRAD jobs are spread out and the users don't get to see anything for a long time. By then half of the users are gone. And they don't even remember what they did to begin with. In RAD you get up quickly. You have something. People are working with something real. As I said before, one of the key things is enticing the users. Unless you have those users, whatever project you're working on, you're in for a disaster. If the users aren't getting what they want, or if they feel it will take forever, they will lose patience. Then they will tell their bosses. Now these people are usually the best managers. They're the people who know how to run your department. You're getting a big favor from their bosses to let them work on the project, because they have taken their best line managers and put them in the project. The line managers aren't happy because people are going to say, "Why do we have them there? We've got work to do. This isn't going to help us. You want to get the users to go along with you. Actually, they can help lead the project.

You keep pace with the competition. RAD is good for those jobs that don’t have a long shelf life. If your objective is to prototype, RAD does work very well in prototyping environments.

The RAD development team has to be small, experienced, and thick-skinned. Thick skinned because if you don't like what they have, they have to go back and redo it. They can't have a lot of pride of authorship. They have to get the requirements from the user, put it up, and see how the users like it. They have to be tough and hearty, because there's a lot of burnout with RAD. This is very intense. We weren't involved the entire time during this RAD project. We were there for maybe two months when RAD was going at full speed and we were working very long hours. But everybody liked it, because we could see something coming. The RAD team members should have worked together before. You have to put up things quickly. Everybody has to know each other’s strengths. Team members must have good people skills because they have to draw out the users. If the team members can't hear—and listening is one of the most important skills they need to have—they won’t get anywhere because this is interactive. They need tools skills, because they work with big chunks of code. They are not going to write each step out in one line of code at the time.

I liken it to forming to a Special-Forces insertion team. That's a small, well-rounded group of people who have a special job to do and they do it. The users don't have to be sophisticated, just enthusiastic. You want to draw the information out of them. It helps if they're articulate, but often they aren't. It helps if they know what they're talking about but often they don’t. That's the biggest obstacle we ran into. You're usually working with a new and innovative product, and things are being developed as you go. One way to change things is to bring in a new system. But in this case, the system often defines what's going on. This is a problem with RAD, because you're moving at such a fast pace, if you encounter a problem you don’t have much time to solve that problem. You have to have people who can help you do that quickly. And these people have to be flexible to see the various approaches that can be used.

Applications. As I said before, RAD is great for an agency or marketing type of environment. In fact, one of the companies Dave discussed drew their agents in because it promised them a personal computer and it found out what the agents wanted. The agents were making trips to the home office, which was a good thing. RAD gives you an approach because the users will usually start out slowly. Then they'll recognize the possibilities that are involved. You can't hold them back at that point.

I think one of the best areas where RAD can help is front-end applications because usually in those you're designing screens. That's hard to do on paper, particularly for users who are not sophisticated. RAD is good for new business applications, which combines the front end with the agent. You're getting people to tell you what they want without having to write it down. Reporting is another area where the RAD process works.

Regarding a particular application, if it's innovative RAD is very good. If it's evolving RAD is good. If it has a short shelf life, you don't want to spend much money and much time using TRAD, because the applications isn't going to be around by the time you come out with it. So you have to do it quickly and you have a much better chance doing that if you use RAD.

Ms. Linda M. Lankowski: How do you get the experts? If you're bringing experts in from outside, how do you build in-house experts?

Mr. Fairchild: That's a good question. You definitely need the experts. In our project they came in and hit the ground running.

Mr. Robinson: On our project we had some in-house people who worked with the experts. Within two to three months, they were fully functional developers on the team. And that's how we approached it.

Ms. Lankowski: Will they become the experts on the next RAD project?

Mr. Robinson: They would have except they have all gone on to consulting positions.

Ms. Lankowski: That's part of my concern too. How do you keep that in house?

Mr. Robinson: These people are expensive. You're teaching your people a skill and the more skills the data processing people learn, the more likely that they'll stay. But it's a problem keeping them.

Ms. Lankowski: My second question has to deal with pricing and product development. How does that integrate with the systems development?

Mr. Robinson: Often you have to service the product you come out with. Often the dictators that I mentioned earlier give you a product and say, “OK, the actuaries are going to develop it, and we want you to administer it in two months.” You can't do that. Or you need to develop an illustration system, a front-end application system, and you could do it quite readily in a RAD environment. You can do it quickly. About pricing, I think it's more of a situation where you can get the peripherals done, as opposed to the specific pricing of the job itself.

Mr. Miller: It's interesting because thinking about the pricing activities that you do, it tends to be almost a RAD process. Actuaries actually have been doing this.

Ms. Lankowski: But generally the pricing actuary doesn't talk to the systems people until they've got a specification done or at least part of the specification done. In which case the systems people say, it's not a detailed enough specification.

Mr. Robinson: And two years later you might have something. That's the whole key to RAD. The systems people do talk to the actuaries and the other administrators. You get into that quickly so that you do have something you can take in new business with or get an application in. It's this whole process of writing perfect specifications that we're talking about. If you wait for that, by the time you're done with the project you may not have any sales. Plus, the opportunity is gone.

Ms. Lankowski: If we don't have specifications how do we maintain the system in the future?

Mr. Miller: Well, you develop the specifications as you're going.

Mr. Fairchild: On our project we've hired a technical writer who will come in and actually document the system as we're developing it. That is where the documentation will come from. But yes, if we did not have that person, we would have to go back at some point and fill it in.

Mr. Miller: How many in the audience have been involved with RAD jobs? Would you mind stating your experience with it?

From the Floor: I just want to know, have you talked to any companies who tried RAD and didn't like it? We heard about all the companies that liked it. Which ones didn't care for it?

Mr. Robinson: Well, I have a partial answer to that. In this particular project we got to a phase where we had the more detailed information to do, and Prudential brought in another team. We realized that the first team didn't leave many trails and the documentation wasn't there, so in all of these things you have some pluses and minuses.

Mr. Miller: There are areas even within the same project where RAD makes more sense and where it doesn't. It's not so much, do I like RAD or dislike it? RAD is another tool that I can use.

Mr. Fairchild: I have one comment to make. We discovered that you really need a knowledgeable end-user, whether or not you have specifications and a piece of paper to look at. You need the person who knows what they are talking about. We realized we no longer liked the RAD process when we got into functionality that was much vaguer than the earlier functionality. So we changed our direction and slowed things down and approached it from a more traditional view for those specific instances. We're still doing the RAD process on the illustration side where we have individuals who know the business rules. But on the administration side, we definitely changed our approach.

Mr. Robinson: For example, we're putting in some tax functionality, which was very difficult to do in a RAD program because it has too many formulas. On the other hand, the illustration side is putting in the same information, but they had a very small unit. The data processing people and actuaries talked and developed a good rapport. They knew whether each one was strong or weak. In that circumstance, RAD worked well. As I said, maybe you have to mix and match. Take a particular application, maybe part of a project, or even the entire project and look at it. What fits for that? Then even within that some of the results that Dave found when interviewing these companies, such as using C++, the companies said, well you should never use C++ in this type of environment. You have to use a RAD as a tool. I mean you have to use it as best you can.

From the Floor: I was wondering what a TRAD team looks like. From your example a RAD team has six developers, two users, and four QA testers. The TRAD team includes design people. Does the TRAD side include anyone else?

Mr. Fairchild: You don't necessarily need to add more people from a TRAD side. It will just take that much longer to develop the system. I actually feel we had more developers than a typical RAD situation calls for. Again, there are specific RAD rules that we had to bend, as I'm sure everyone else would. The team wouldn't have been so large had we had all of the expertise that we needed. For instance, QA testers. We needed four because we had little experience there. And we had very little experience in using any automated tools. Again, we had to add in one area, because we were lacking in another.

From the Floor: I think the critical component of what you're doing is trading off managing your functionality, in terms of adding something that might take many months to program that you would never use administratively more than one policy a year. Is that kind of trade-off, in terms of recognizing the fact that it's going to take a long time to program, worth putting in to get it in this 90-day time frame? It helps a lot in terms of savings and cost on the project. However, in the system that you're producing, is there that much reduction of functionality? In terms of efficiencies, is the system less efficient or more efficient when it's done?

Mr. Robinson: I think we do a lot of cost benefit thinking. And if we don't think the application is going to be used much, then we avoid it. If it’s something that's critical then we do it. If it isn't critical and the chance of it is rare, then we probably don't do it. This leads me to something else that I want to say. You can't do RAD testing. You have to do testing in RAD, but you can't do RAD testing. In the old days people would just try an application as a means of testing. You still have to have a good test plan in RAD, and you have to accept the things as you would in TRAD. However, even in TRAD we decided not to test everything. People said, “Well, if it's in the system we should test it.” We're not really doing that. I don't think it is cost-effective.

From the Floor: You keep referring to RAD tools, and you mentioned that C+ + is not one of them. What are the RAD tools?

Mr. Fairchild: RAD tools are those tools that will generate much of the background code for you. Power Builder happens to be one of them. The developer will do some coding and some screen design. But the tool itself will generate much of the background code. SQL code is needed so use Power Builder because it is capable of generating a lot of code. C+ + is not considered a RAD tool.

Mr. Miller: I think it's important to differentiate the RAD approach from the RAD tools. The RAD tools can enhance your RAD approach. But I talked to two people who said they've done the RAD approach using COBOL. It's possible but you have to have the right elements in place.

Mr. Robinson: Report writers and things that allow you to build large chunks quickly are the tools that you can use. I've seen COBOL code generators that produce large chunks of code quickly. You need reusable things too. You have to take things and set them in quickly to keep the attention of the users. You have to keep moving.

Mr. Miller: Does anyone want to share their RAD experience—whether positive or negative?

From the Floor: I'd be willing to share my experience, although it's with an illustration system. I work for an illustration software company. And we've always believed that you really only need three people to work on a project. You need a tester, a programmer, and a coordinator. I think that it has served us well, in terms of limiting the functionality and comparing the cost of something to the time it takes to get it in. I think it gives the company a focus, and saves it a lot of money, and gets the systems quickly. I'm surprised to hear it doesn't scale too much to the larger systems. The illustrations have worked wonderfully. I think for many of the systems we've had great success with it. Although that's in retrospect because we really didn't plan on using RAD.