

RAD REALITIES: BEYOND THE HYPE TO HOW RAD REALLY WORKS

Building what the customer needs now and delivering it quickly is the new rallying cry for software development

by Ellen Gottesdiener

The volatile nature of business in the 1990s along with the emergence of newer technologies such as client/server, object-oriented (OO) technologies, and an ever growing list of application development tools has made rapid application development (RAD) a buzzword that falls off the lips of almost every I/S manager and developer. RAD has been described as a tool, a methodology and an attitude. It is associated with prototyping and techniques such as joint application design (JAD), which require customer involvement and commitment. It is also feared as potentially a source of low-quality applications that do not scale up or that cannot sustain the demands of ongoing maintenance.

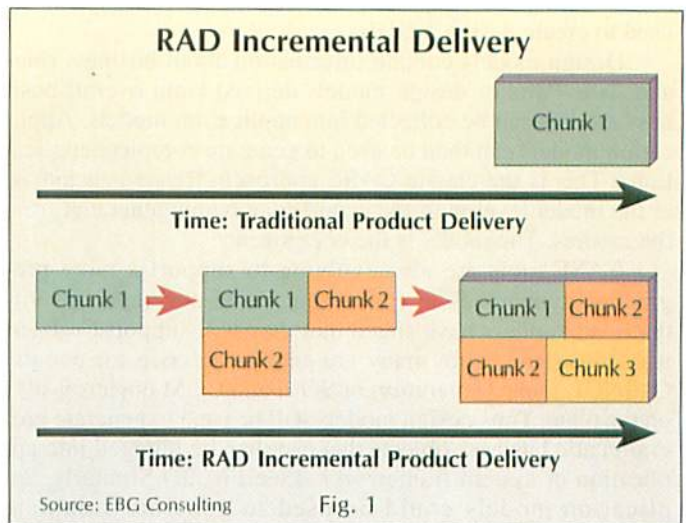
The questions beg to be asked: "Is RAD just another I/S fad? How might this approach to application development change your life as a project manager or developer? What can the roots of RAD reveal to us about the essence of this 10-year old approach? Can it live up to the pervasive marketing campaigns of tools vendors? What must be done to succeed with a RAD approach?"

DEFINING RAD

RAD is an integrated set of techniques, guidelines and tools that facilitate deploying a customer's software needs within a short period of time. This predefined timeframe is called a "timebox." The software product does not "pop out" at the end of the development cycle, but instead evolves during the RAD development process based on continued customer feedback. In addition, the whole software product is not delivered at once, but is delivered in pieces by order of business importance. In this way, the product is deployed in increments over time. (See Fig. 1.)

Each increment, or "chunk" of the application typically takes three to nine months. The RAD process

requires a small team of highly skilled individuals (including customers) to work together using tools that accelerate the testing, prototyping and construction artifacts of the software product. It should be noted, however, that there can be exceptions to the small team size, particularly when dramatically new technologies



RAD products evolve based on continual customer feedback and are deployed in increments over time.

and techniques are employed. (See "Cigna Develops One-Year Health Benefit System," page 35.)

PLATFORM INDEPENDENCE

RAD techniques and tools are platform independent. RAD is not an excuse for "hacking" out software. For the PC developer applying RAD, more discipline is needed. For the mainframe developer applying RAD, more flexibility is needed. The key is striking the right balance.

The software product development process is very different than traditional application development. Rather than focusing on steps, tasks, and interim deliverables, RAD is oriented toward product delivery cycles defined by an end date. Thus, the successful delivery of

Ellen Gottesdiener is president of EBG Consulting, Inc., a Carmel, Ind.-based facilitation, consulting and training firm specializing in helping organizations create usable business and technical models for information systems. She can be reached via E-mail at eg@compuserve.com.

a product is based on the skills and ability of the team to learn about customer requirements as the software product is being built. In an evolutionary process, learning and adapting to the environment is key. By exposing the software product to the environment quickly and by letting customers critique, review, and provide feedback on a prototype, the team can make changes to the software to allow for better adaptation to the future production environment. The product evolves within the timebox while the team learns. In addition, the quality of the application is dependent upon maintaining a stable "memory" of the product structure. This is accomplished through the use of models underlying the interface design.

RAD ROOTS

RAD emerged not from the PC or client/server world but from the midrange Digital Equipment Corp. VAX environment in which Scott Shultz, then a project manager at DuPont, used Cortex's CorVision code generator tool in conjunction with new development techniques. Shultz called it rapid iterative production prototyping (Ripp). Ripp involved uniquely combined tools, methods and people to deliver systems quickly to customers. His approach was later popularized by James Martin's 1991 book, *Rapid Application Development*.

Now a senior manager at Ernst & Young LLP in Dallas and having participated in over 100 RAD projects in his career, Schultz's approach was incorporated into Ernst & Young LLP's Navigator methodology to form the Accelerated Systems Development (ASD) route map. ASD, says Shultz, is essentially the "grandson of RIPP." Shultz's colleague during RIPP's early days at DuPont, Bucky Wallace, adapted similar techniques to the MVS mainframe environment. Working in parallel with Shultz, Wallace formed a technology group called Application Systems with Accelerated Productivity (ASAP) within DuPont. ASAP used mainframe tools with a traditional waterfall methodology. Starting with the goal of increasing productivity by tenfold, Wallace discovered that even if the code generation portion of the lifecycle was reduced to 0%, overall savings in time only came to 30%. "I discovered that you have to change the way you do things — the tools were not enough," said Wallace, now President of ASAP Systems Inc., Landenberg, Pa. "Tools don't solve problems, people solve problems," he added.

BUSINESS PROBLEMS FIRST

Wallace emphasizes in his RAD approach the ne-

cessity of focusing on the business problem first (See Fig. 2.) Other RAD practitioners began by using tools in a mainframe environment while also changing the development methodology. Rob Dixon, a partner with Tier Corp. of Walnut Creek, Calif., and author of *Winning with CASE*, used the IEF CASE tool from Texas Instruments, Plano, Texas, in a Cobol/DB2 project. Dr. Sam Bayer, consultant at Sapiens USA, Inc., Durham, N.C., used Sapien's own tools in mainframe RAD projects. — A detailed accounting of RAD in the mainframe environment is told in Kerr and Hunter's book *Inside RAD*, a book that credits Shultz for his early work at DuPont.

Although tools are very critical to the RAD approach, the tools alone will not create a stable, scalable software product. "The tools have taken over the team," said Shultz. It is critical to look beyond marketing to the people, process, and organizational issues to make RAD what its inventor envisioned it to be.

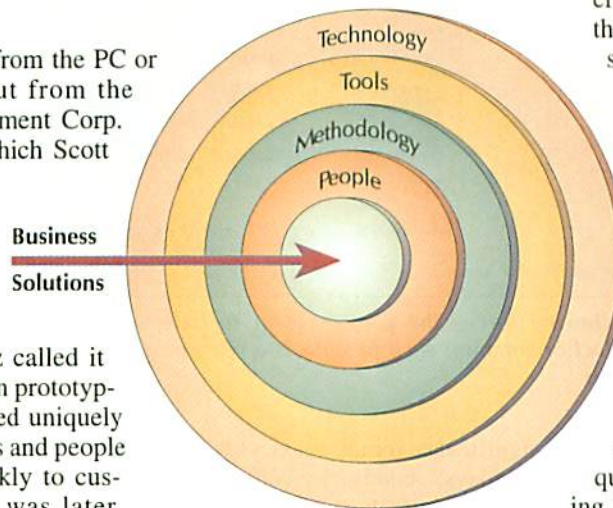
STEPS IN RAD

The RAD process defies a linear definition of steps carried out in a sequence. RAD begins by defining the desired product in an initial planning phase. During planning, a definition of the project scope is completed along with some preliminary data/process analysis, risk assessment

and estimating. A timebox is chosen — a fixed period for building a fragment, increment or chunk of the application. Within this timebox, a spiral process occurs involving prototyping, modeling, architectural design, construction and testing. (See Fig. 3.). Each cycle within the timebox is completed "multiple times, each time bringing the solution to higher levels of sophistication and completion," said Shultz of Ernst & Young. In some cases, a higher level of analysis may be needed to insure proper prioritization of RAD projects. An information engineering-like analysis may then proceed these steps. (See "USA Group Spices up Scholarships," page 32.)

Rob Dixon, Tier Corporation, described conducting a one-month business area analysis (BAA) to determine the subsets of application development needs in the whole business area and divide them into chunks of requirements that can each be fulfilled in four to nine months of development. During this one month BAA, the team creates models to define the business context, including data, functional decomposition, and perhaps object models with the aid of Case tools.

Bull's-Eye for RAD



Source: ASAP Systems Fig. 2

The primary focus in RAD should be the business problem, not the technology.

"We take the most compelling ones and do them in the first chunk," said Dixon. Early and rapid definition of the project means understanding the business problem, assessing the risks, prioritizing the needs, and defining the increments in order of development. These early activities have the affect of team-building as they focus sharply on business needs and product delivery. It also does much to gain and maintain customer involvement, a key aspect of the RAD process.

Applying more discipline and less art (and guesswork) to RAD planning has been successful for RDI Software Technologies, Des Plaines, Ill. RDI uses a well-defined process for sizing and estimating projects, according to Todd Wyder, company methodologist. RDI uses metrics, including function points and quality function deploy (QFD), a series of customer-oriented ranking techniques rating, software complexity and risk.

After an initial JAD session to determine scope by defining inputs and output, RDI works with customers to determine application chunks by defining "use cases" — an event-driven analysis technique invented by Ivor Jacobson. Each chunk is assigned a customer-risk-complexity factor, and the whole effort is estimated using function points.

For RDI this technique is objective, repeatable and understandable. Customers "buy" project components based on the cost of the function point. When changes occur, as they will, function points of the requested change are determined to estimate the cost and time needed for the alteration. Customers can then decide if they want to buy the modification. "We take nine months to one year to get 12 to 15 deliverables in order of importance to the customer, so it's very exciting to the customer," said Wyder of RDI. "The higher risk and complexity components will be delivered early in the project."

TIMEBOXING

Timeboxing is an essential project management aspect of RAD. It forces the team to anticipate reducing the scale of product delivery, requires focus on

customer priorities, assumes continual change will occur and imparts to the team a sense of urgency.

A timebox is a mechanism to control resources and delivery scope; it provides a dramatically different way to manage a project. "The timebox is based on the belief that we can do something of

at a common goal, scope and size. The development team and customers together determine the project's timeboxes. According to Highsmith, "The team has to be able to juggle features to meet the time that the customers help to decide. The customers know that any feature-set planned for the last cycle has the potential

to be deleted if the overall timebox is in jeopardy. Timeboxing doesn't work unless both developers and customers understand the need for trade-offs."

The focus group is facilitated to de-politicize and de-emotionalize diverging views. It is called a focus group to underscore the intent and origins of the concept. "Focus groups work so well because customers are given things to do that they understand. They enjoy and understand and get positive feedback from criticizing developers," said Bayer of Sapiens. He recommends that the first timebox should be delivered in three or four

weeks of project initiation to validate the data model. A menu structure is thus delivered that demonstrates the cruding (create, read, update and delete) of the major entities.

According to Bayer, "this first version also establishes a true partnership with the customer. The product is ugly, but it is also kind of in their language. They are still skeptical because they can't use it until other versions are delivered that capture and implement the business rules and their workflow. But at each focus group, they are given the opportunity to criticize the application without criticism of their criticism."

Shultz suggests such other options as: put it on a wish list, add resources, substitute other items or just do not do it.

The timebox concept is thus a management control device, forcing the team to maintain and manage scope.

TIMEBOX ITERATIONS

Release, version, feature set, increment, cycle, fragment, phase, segment, build, cluster, use case, chunk, all are terms that mean the same thing. They are a subset of usable functionality that is delivered to the customer at a predetermined point in time. Chunks are created

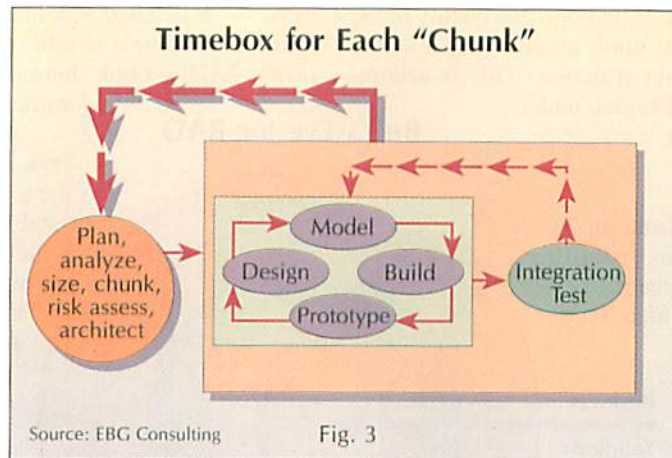


Fig. 3

Timeboxing forces the project team to have a market and product orientation; each timebox is a release or version of the system.

significance for the business within 90 to 120 days," said Bucky Wallace of ASAP. "Multiple timeboxes can go on sequentially, in parallel or staggered, and are linked together by project management and data management. In this way, we develop systems the way they are supported; each timebox is a release, or version of the system." (See Fig. 4.)

Timeboxing forces the project team to have a market and product orientation. Planning product components by timebox, rather than tasks and activities by deadline, is an effective means of delivering real value. Jim Highsmith, principal of Knowledge Structures Inc., Salt Lake City, Utah, collaborated with Sam Bayer of Sapiens to devise a RAD approach that they named Radical Software Development, which includes the use of "customer focus groups." The customer focus group, a concept borrowed from consumer marketing, is a team of knowledgeable customers that gather in JAD-like sessions to review an implemented portion, or "feature set," of the application.

JAD SESSIONS

The Radical Software Development cycle begins with a JAD session to arrive

USA Group Spices Up Scholarships

RAD effort incorporated E&Y's Navigator, Sterling's ADW, CCI's tool, training

It could be likened to a tortilla chip whipping through a batch of spicy dip. Team Salsa is the Fishers, Ind.-based USA Group's client/server development group for scholarship and loan systems administration. Team Salsa used a hot-pursuit methodology to deliver the first increment of a new application in three and one-half months. Informally known as Salsa, the completed application will process loan and scholarship applications for private institutions, clients and associations.

The application is divided into phases and timed to correspond with the actual cycle of scholarship and loan processing.

The application's first phase processes client-supplied demographic data, creation of the application packages for potential applicants, application entry and edit, an interface to facilitate phone customer assistance and call history tracking.

The second phase, delivered in five months, processes ranking and scoring the applications and correspondence to all applicants — approved or not-approved. The final phase, to be delivered in three months, will process funds management and fund reconciliation. Thus, each deployment of the system builds on a previous version, thereby forming an incremental software development process.

The delivery dates for each timebox increment matches the real world school year loan processing cycles. The application also supports USA Group's scholarship management business area strategy, which includes such goals as compliance with internal audit reporting, provision of enhanced service options, and increasing of USA Group's market share.

The Salsa application is used by seven staff members in the scholarship management group. It uses a two-tier client/server architecture, with Windows-based clients running PowerBuilder from Powersoft, Concord, Mass., against the server, which uses the Sybase System 10 database engine operating under Unix on an HP9000. The Sybase database is comprised of 22 tables with some 300 attributes.

Database connectivity is accomplished with Sybase's DBLib and CTLib middleware running on TCP/IP. Prior to Salsa, the application consisted of a combination of manual processes, Lotus spreadsheets and Paradox databases.

The project began with a small, well-trained team of five: project leader Dawn Denman, two developers, a testing analyst and Jon Jones, a technical consultant. Additional I/S resources, such as database administration and security, assisted the team on an as-needed basis. The end

users, the seven loan analysts, were always available as adjunct team members.

The GUI developers had previously been trained in PowerBuilder and had used it to develop another single-user application. USA Group had Navigator Series from Ernst & Young, Irving, Texas, as a methodology, and the team had experience using the information engineering path of Navigator and the ADW CASE tool from Atlanta-based KnowledgeWare Inc. (Now owned by Sterling Software.) So Denman decided to "pick a recipe" from the Navigator methodology for the Salsa project.

Beginning with a scholarship management business area requirements analysis (BRA) activity that took three weeks, the team conducted a Salsa business analysis in three weeks, conceptual systems design in one week, physical design in 10 days, iterative prototyping in two weeks, and development and deployment in six weeks. Denman set the timeline for the phases herself to fit within the overall three and one-half month timebox.

As part of analysis and design, the team created models of the structure and activities needed in the application. Models built during business analysis included: a context diagram that had 27 data flows, a process decomposition diagram with 19 elemen-

tary processes, business transaction process descriptions and an entity relationship diagram (ERD).

Conceptual systems design entailed converting the elementary processes into business transactions and creating process descriptions describing the triggers, logic flows, and inputs and outputs. This work was done with extensive end-user involvement. "The best user involvement I've ever had," said Denman. End users reviewed and critiqued all the models generated in ADW.

The physical design, completed in 10 days, converted the logical design into the physical database schema and the design for the stored procedures. The ERD stored in ADW was used to create the physical schema.

Quantitative information, such as the number of transactions and records expected, was used to decide if a process should be allocated to the server (to be executed as stored procedures) and not to the client (and run as PowerScript code).

The Salsa team immediately followed these design activities with iterative prototyping in PowerBuilder. In this two-week period, the team designed, constructed, and tested the user interfaces.

This entailed two iterations of the interface. After the first version was reviewed by the end users, changes and is-

within a timebox and can be done sequentially, concurrently, or overlapped with each other. (See Fig. 4.) A chunk can be prioritized based on business need, dependencies and/or risk. For example, a risk-based approach involves creating a wide navigational prototype — as opposed to a deep prototype — to validate the data model, to discover the overall customer workflow or to test the feasibility of new technologies before proceeding with the other components of

the RAD effort.

Anecdotal evidence supports the notion that three iterations are needed to complete a chunk, although variations can and do exist. An iteration can be measured in time either by number of iterations or by more concrete metrics such as function points. The process is likened to filling up a shot glass as opposed to filling up a water glass, said Bucky Wallace. "You pour water into the shot glass, implementing hunks of dis-

crete functionality instead of trying to fill up the water glass at once."

ARCHITECTURE

Determining what is a chunk and the order of creating them occurs in the RAD planning step. Certain chunks depend upon other chunks. For example, an order fulfillment chunk depends upon the order entry process to create the "order" object and the order entry process in turn depends on the pricing process to

sues were documented and the team created another prototype. In some instances, the team created two samples of an interface so customers could choose between them. A total of 42 dialog reports and eight paper reports were designed.

This phase was possible within two weeks because a significant skills and knowledge curve had already been overcome. USA Group had previously purchased the GUI Guidelines product from Corporate Computing International, Bannockburn, Ill., and the designers had attended Corporate Computing's effective GUI design training.

Moreover, through working closely together on the business models earlier, a high level of trust between the developers and the users had developed by this point in the project.

A few of the technical difficulties the team encountered are being addressed for the last increment of the application, according to Jones, who served as the technical lead on the project. There were problems getting some of the PowerBuilder extended attributes in the physical schema generated by ADW, so Jones is looking at other tools such as ERWin from Logic Works Inc., Princeton, N.J., as the bridge between logical and physical database design. Sharing a single user version of ADW also became cumbersome at times. Although team members had some basic SQL training, it was noted that extensive training on Transact SQL and Sybase stored procedures would have helped.

Denman warned other project leaders not to attempt RAD without absolute user

commitment and involvement. In addition, it would not be possible to accomplish RAD for a business application that is not well known and understood. She noted that management buy-in to RAD goes beyond project approval. "It means removing obstacles," said Denman.

Although the team had concerns about the time frame, it worked together with the customers to meet the deadline.

At the end of the project, when a LAN crash disaster caused five days of lost work, the team had to pull some significant overtime to meet the date. Another significant aspect of successful RAD is keeping the team small. "There's been a lot of media hype about flat team structures," said Denman, who found that team rapport is essential in making RAD work.

Denman also commented that some deliverables "can't always be pretty," because of the constraints of timeboxing.

A final piece of advice to RAD hopefuls: be sure your technical infrastructure is already in place before the time-box begins. This includes establishing the workstations and network, the database engine, GUI guidelines/style guides, and basic training.

RAD is more of a "mind set," said Denman. It requires using a flat organizational structure, using GUI development tools and streamlining the traditional application development methodology. With these ingredients on the table, the Salsa team has quickly created an application, filling a real need for USA Group, like a chip through dip. □

— Ellen Gottesdiener

MASTER THE ART OF SOFTWARE TESTING



AUTOMATOR QACENTER™

THE TOTAL SOLUTION FOR ENTERPRISE-WIDE SOFTWARE TESTING

Test Planning
QAPLAN™



Problem Tracking
QATRACK™

Test Script Automation
QARUN™



Stress Testing
QASTRESS™

1-800-486-7565

AUTOMATOR™



Automator, Automator QA, QACenter, QAPlan, QATRack, QARun and QA Stress are trademarks of Direct Technology Limited in the United States, UK, and/or other countries. Microsoft and Windows are either trademarks or registered trademarks of Microsoft Corporation. Powersoft and PowerBuilder are trademarks of Powersoft Corporation. Other brands or products mentioned herein are trademarks of their respective holders and should be treated as such.

Direct Technology Ltd. 10 East 21st Street, New York, NY 10010 (212) 475-2747 Fax (212) 529-4941

CIRCLE #33 ON READER SERVICE CARD

create the order. This implies that early in the RAD cycle, architectural design must be completed to discover these dependencies. "You need a good layered architecture and concurrent development techniques," said Christine Comaford, whose company, Corporate Computing International, Bannockburn, Ill., provides a process management tool called RadPath. Corporate Computing recommends an architectural layering made up of GUI, application logic, business rules and database access.

MODEL DESIGN

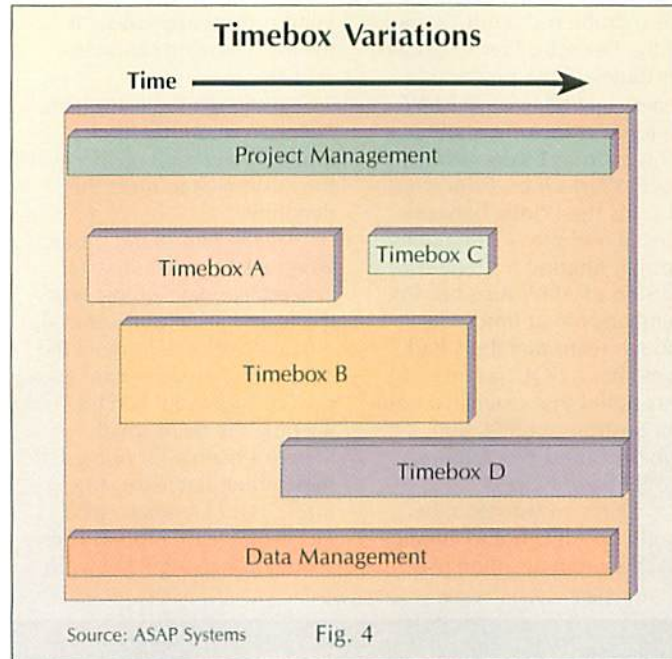
A solid application architecture is based on well-designed models of the data, process or objects. Traditionally, these models are built in-house from scratch by the application team as they build the software. Alternatively, they may be part of the software modules when a third-party software package is purchased. Recent research shows a growing trend in purchasing design templates — workable software built from design models with a CASE tools that can be modified at the design level for the needs of the organization. This has been explained by Hofman and Rockart in the article "Application Templates: Faster, Better, and Cheaper Systems," in the Fall 1994 issue of *Sloan Management Review*.

An existing design is thus customized, saving the RAD team from reinventing the models from scratch. Sam Bayer from Sapiens experienced the combined speed of RAD and power of a template style architecture in one consulting engagement. He worked on an insurance application with an architectural template created by a consortium of 40 insurance companies and now marketed by IBM as the Insurance Application Architecture (IAA). The complete template contains some 170 entities, their relationships, and processes to support some 500 business functions.

On the project Bayer was involved in, the team used a subset of templates to build a working software application.

The successful development of the application used a combination of RAD techniques and the design templates from the insurance architecture models. Having the application architecture of a RAD

edge Structures. "That is why we are seeing an increase in the number of data warehouse projects, namely to permit disparate databases and eliminate islands of automation." Similar danger may be lurking for enterprises that engage in RAD efforts without considering how and if the application architecture will fit with the rest of the technology.



Multiple timeboxes can go on sequentially in parallel or staggered and are linked together by project and data management.

project fit into the overall, enterprise-wide technical architecture can be a challenge, however. In some cases, the enterprise architecture needs to be in place before any RAD efforts can begin.

A solid application architecture is based on well-designed models of the data, process or objects.

"Prior to building a significant number of RAD projects, it is necessary to do an architectural definition that includes tools, databases and implementation architecture," said Highsmith of Knowl-

ENABLEMENT TOOLS

Tools are a critical part of the RAD process. Beyond the typical tools associated with RAD — those that build the GUI interface, for example — are tools that capture the application models, document the development process, create test scripts enabling the whole testing process, generate and build code and assist in software configuration management. I/S organizations may be misled or misunderstand the role of tools.

A good example of RAD practitioners who use a variety of power tools is RDI Software Technologies. The company uses all of the following:

- Rational Rose from Rational Software Corp., Santa Clara, Calif., for defining project object models;
- Lotus Notes from Revelation Technologies, Stamford, Conn., for all documentation, which also permits customers to get access to all project data;
- Interface development tools such as FoxPro, C++ and Visual Basic from Microsoft, PowerBuilder from PowerSoft, Concord, Mass.
- SourceSafe from Microsoft for configuration control;
- QAPartner from Segue Software, Newton Centre, Mass., for regression testing and automated script building;
- PC Lint from Gimpel Software, Collegeville, Pa., for C++ error checking;
- BoundsChecker from Nu-Mega Technologies, Nashua, N.H., for C++ memory leakage; and
- AppPolish from Encore Computer Corp., Ft. Lauderdale, Fla., for interface design spell, button and control key checking.

MEASURING RAD

There is no industry standard by

Cigna Develops One-Year Health Benefit System

RAD team delivered using Digitalk's Smalltalk, OTI's Envy, ObjectShare's GUI Tool

Imagine a group of high level I/S managers attending a five-day course on Smalltalk as a means to understanding object technology and tools. That event marked the commitment by management of Cigna Corp.'s Cigna Health-Care division in Bloomfield, Conn., to commission a one-year R&D effort to develop a benefit information decision system. The project began in 1992, after having been attempted several times before within the division.

In the latest attempt, according to Alan Kirk, lead architect, Cigna HealthCare, the project took a completely different form. The application development effort in the latest approach included a business process reengineering (BPR) effort. Cigna developed a system that enabled it to clearly define benefits packages for any client purchasing a Cigna health care product.

The benefit information decision system application, created within a one-year timebox, was deployed in one of the six Cigna managed care service centers, each of which has 30 to 100 users. After the one-year RAD effort, the RAD project was recommissioned. The team rethought the cycle time for delivery and decided that the focus should be on six-month deliverables.

This accelerated delivery schedule is now being used to roll-out new and/or redesigned functionality to the application. Currently, the third evolution of the application is in development by 14 project team members. The application allows Cigna's service centers to create, modify, share and report on health care product benefit information. The first application had three business functions: product engineering, benefit plan sculpting and inquiry. It was built in collaboration with a consulting firm, Symetrix, of Burlington, Mass.

A team of 12 to 18 team members built the system, and during one two-month period, the project team grew to as many as 35 members. The consultant provided project management and half of the developers. Half of the total team were business analysts. The effort continues with all Cigna employees located in Bloomfield.

The application interface and business logic was built using Smalltalk from Digitalk Inc., Santa Ana, Calif. The team wrote routines to store persistent data in Sybase as binary large objects (Blobs) in image columns. Each object has its own row. Envy from Ottawa-based Object Technology International Inc., also used by developers, permits partitioning an object's methods to different application components, thereby permitting their extension and reuse. The team also used WindowBuilder from ObjectShare Systems Inc., Santa Clara, Calif., for building user interfaces. A two-tier architecture was used in the first version with OS/2 clients running Smalltalk and the regional server running Sybase under OS/2.

The process model for delivering benefit information application was an evolutionary one. The first release of the software product was deployed in "the real world." Improvements were needed and recognized, and subsequent versions of the whole application have been deployed. Each version has been reshaped in both form and functionality from the prior version. Six months after the first version was deployed, 80% of the application had been rewritten, and additional functionality had been added. The rewritten portion included redesigning objects, a step that permitted the end users to more efficiently and easily drill into benefit plans.

The method for building the system was very different from any method previously used. The team would "chunk-up a development effort and we would assign one or more analysts, developers, and users," according to Kirk, who was the lead architect for the effort.

For each chunk, the team tools a portion of the interface and returns it to the rest of the development team to see how it integrated with the rest of the chunks. The team chose to work with the users and show them the interface and "let it settle for a while."

Different team members had different preferences for when they would share their "chunk." Some, like Kirk,

shared their plan and design before coding while others waited until after it had been coded.

Smalltalk, as the development tool, permitted rapid rewriting of all or subtle parts of the application. Developers can incrementally add functionality to Smalltalk objects to make them richer in behavior. The RAD approach goes beyond mere prototyping, according to Kirk. "You build to implement and show the interface to the client. We typically have to write things three times before we get it right," he said. Kirk also noted that object development enables modeling of the business with flexibility and "provides a strong methodology to model

COMING SOON
TO ENLIGHTENED ENTERPRISES EVERYWHERE!

**COMMANDER
UNIFACE**



THE
MOST ROBUST
AND ADAPTABLE
CLIENT/SERVER
APPLICATION-BUILDING
STRONGWARE IN ALL GALAXY-DOM! A COMPUWARE PRODUCTION!

For a free color episode of Commander Uniface quelling an outbreak of information stagnation on the Hamidic planet of Drago, call 800 365-3608 or catch us on the "net" at uniface_info@compuware.com



COMPUWARE
Uncomplicating Your Life

Compuware and UNIFACE are registered trademarks of Compuware Corporation.
© 1995 Compuware Corporation

CIRCLE #35 ON READER SERVICE CARD

and build; it is a faster development approach."

For the first effort, objects were modeled with paper and pencil. The team also used a bit of class/responsibilities/collaboration (CRC) modeling, an object methodology designed by Rebecca Wirfs-Brock. Currently, Kirk is investigating object-modeling CASE tools to accelerate object analysis and design and for storing models required to cross-train new team members.

The "peopleware" aspects of the project were a significant factor for success. Team decision making was collaborative, not directive. No one person was in charge, although there was a designated project manager. That manager let the team, as a whole, prioritize what would be delivered and when it would be delivered. The team set up two-week delivery cycles.

"It ran like a Swiss watch," said Kirk. "Every two

weeks we'd plan the next deliverable, always focusing on the most important pieces within those two weeks." The team held daily morning meetings to review issues, and it also held a longer status meeting on Monday mornings.

The RAD effort required significant education in both methods and technology. Skills, such as building an effective GUI, working in a PC environment, and the biggest hurdle, understanding objects, required learning. Among the greatest learning challenges were such new concepts as class hierarchies, order collection, containers, arrays and dictionaries.

"They use the term 'new paradigm' and it's really true," said Kirk. "It takes longer than you want to admit," he added,

"perhaps three months to be competent. But you're not an expert until you've had 18 months or maybe three years of experience."

In addition to using the object technology, analyzing and designing business objects sometimes brought interesting and unexpected surprises. Subtle, hidden objects emerged, which did not appear during initial analysis.

For example, said Kirk, without realizing it, "clients make choices about the product that were not verbally expressed during object modeling," but were caught once the client had its hands on the application.

Kirk outlined critical success factors for both the initial and current benefit information decision system team: good tools; user buy-in; un-

derstanding by the users that there will be sacrifices in the name of speed; and time after deployment to perform rework.

Kirk noted that at times the systems team members would create "spaghetti" to get something out quickly and then it would be necessary to go back to the core technology to rework it with a better technical foundation. However, according to Kirk, the time to perform that rework was outweighed by the value of speedy delivery — as long as time was taken subsequently for technical cleanup.

Reflecting on the first one-year RAD effort, Kirk said the team erred on the side of speed. If one were to imagine a continuum with analysis paralysis at one end and hacking at the other, they would have spent more time on analysis. □

— Ellen Gottesdiener



Alan Kirk
Cigna

which to measure software, although function points seems to be a de facto standard or foundation for capturing productivity and defects in software. Some RAD metrics factor in other elements such as skill levels of individuals, number of dialogs, events, risk factors, and degrees of documentation. Metrics for RAD are relatively new.

As a rule of thumb, according to Rob Dixon, Tier Corporation, seven- to nine-month RAD efforts typically entail 30 entity types. Highsmith of Knowledge Structures, says that each RAD project tends to be less than 1,000 to 1,200 function points. Comaford of Corporate Computing, finds that an individual with beginning skills can produce 20 "work units" a month — using a modified form of function points in the Rad-Path estimator tool — increasing to 50 and then 80 works units for intermediate and advanced individuals.

Capers Jones, metrics guru and president of Software Productivity Research, Burlington, Mass., pointed out that RAD has declining value on large projects, and thus it is best for efforts sized at up to 500 or even 1,000 function points. His research indicates that fewer than 5% of all commercial software

shops are using RAD. That may change in time. "When something becomes popular, it becomes explosively popular," said Jones.

Another perspective is provided by RAD's originator, Scott Shultz, who points out that function points are deceiving, because the tools automatically generate some of the artifacts that are counted in function point analysis. "If we look past the issue of productivity — when we once looked to save a few weeks time — the critical path is now users who have to transfer information, make decisions and prepare for and manage implementation of the software product. We have been measuring the wrong things in the RAD environment. We should be focused on the productivity of the user's decision-making and change management," said Shultz.

TESTING C/S RAD

The benefit of applying RAD to client/server development is not yet proven, according to Capers Jones. "Client/server has its own set of problems, such as horrendous quality," he warned. "The industry standard is three defects per function point, while for client/server it is four defects per func-

tion point. The best prevention steps are manual inspections and testing."

Jones' research indicates that inspection is more than 95% efficient in defect removal. "The best prospect for us is the team that has built one client/server application and tested it manually," said Eric Schurr, vice president of product management and marketing for SQA, whose company offers a GUI client/server testing tool and methodology called SQA TeamTest. "Now they know how important and difficult it is to test a client/server application. The only way to test it properly is through automation. In addition, you can't wait to the end of the RAD project to do a test," said Schurr.

Testing provides quality-related metrics, such as defects by delivery cycle and priority of defect correction. Project defect tracking "moves project management out of the realm of feeling into the realm of fact" and can assist in measuring application quality by identifying, categorizing and planning how to fix bugs. In addition to testing tools, testing as a lifecycle process is built into RDI Software Technology's RAD approach. This is accomplished by defining and involving a testing role from the beginning

of each project. RDI assigns a quality assurance (QA) role to each project and an additional technical QA person to larger projects. These individuals are not developers, because as Todd Wyder pointed out, "The tester is into breaking and the developer is into making."

SKILLS/ROLES

The RAD process requires skilled individuals to play one or more roles and requires increased knowledge of both business and technology. (See "Relearning: The Keystone for I/S Technology Migration," *A/D Trends*, December 1994.) "We need 'renaissance people'," said Shultz, "people who know a bit about everything, but we also need specialists like database administrators." I/S typically provides the people to perform such roles as facilitator, applications architect, GUI designers/developers, application developers and QA testers. An infrastructure group may dip in and out to assist the core project by providing expertise in database administration, networking, prototype design and architectural design.

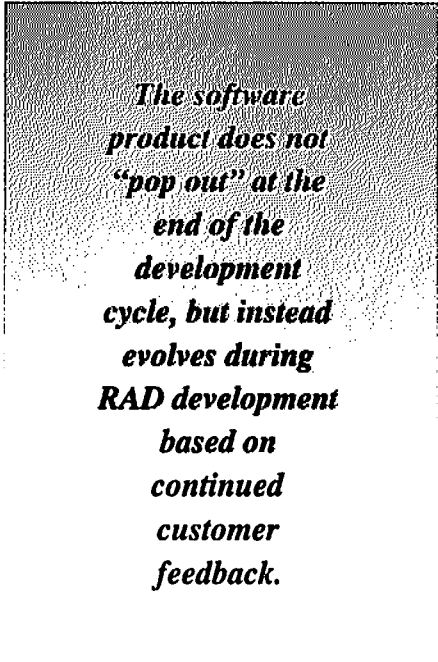
Whatever project organization is used, it must facilitate RAD's need for high performance work teams. Not all members can know all the tools, methods, standards, libraries and technologies. That is why roles are necessary. Each individual would ideally play multiple roles, for example, application architect and data administrator. Multiple roles played by several developers may also be necessary, and roles should be assigned to critical players and to cover important bases.

CUSTOMER ROLES

Most critically, customers must provide several roles in a RAD project and be active team members. An executive sponsor must be identified who approves the project and permits the business analysts the authority to approve the deployment of a RAD increment. Business analysts will be part of the project team defining data, business rules and workflow and also critiquing the interface prototypes. In addition, the team needs to have the right kind of users — not just business experts but also ones "who are willing to not haggle over added functionality within the context of that phase," pointed out Marie Benesh, division I/S manager at Corning Inc., Corn-

ing, N.Y.

Benesh participated in a RAD project last year. People on the RAD team, typically four to seven people, must learn to behave differently. Traditional I/S roles must be played with great flexibility. For example, a database administrator may be asked to create and drop tables many times in a RAD project, to accommodate multiple iterations and multiple increments of the software product, instead of once or twice as in a



The software product does not "pop out" at the end of the development cycle, but instead evolves during RAD development based on continued customer feedback.

traditional I/S project.

The project manager must place less emphasis on plans, tasks and deliverables and more on problem anticipation and immediate resolution of issues that would traditionally take days, weeks, or months to resolve. The project manager is a cheerleader, who excels in communication skills, has a deep appreciation of the technical details and should not be, as Sam Bayer said, "retentive about the plan." Finally, all team members must be "comfortable with the ambiguity," pointed out Benesh.

THE RAD ATTITUDE

"Evolution is chaos with feedback," said physicist Joseph Ford. A RAD effort is chaos bounded by the timeboxes, and it is dependent upon the continual feedback of the team itself. Customer team members provide feedback on prototypes. Analysts provide feedback on the models. An even more evolved RAD

process means continual feedback on the RAD process itself. This self-examination, pointed out by Todd Wyder at RDI, involves the ability to "kick back to think about what you did; stop at regular points and see what we are doing well, not so well, and what are we going to do about it — how can we change that. That will help the incremental process improvement all the time."

Even one-hour evaluations every other week can do much to help the process evolve, along with the product. RDI also always conducts a "postpartem" check for everyone else in the firm to learn from the project. All evaluation sessions are recorded in Lotus Notes from which everyone in the firm can review and learn. RAD uses tools and techniques that are platform-independent. It requires striking a delicate balance between the hacking associated with the PC development environment and the analysis paralysis typically associated with waterfall methods.

In addition to using such techniques as timeboxing, chunking, customer-driven product delivery and high performance teams with high performance tools, RAD is based on the premise that software development is a discovery process. Efficient discovery is complex. It involves risk and uncertainty mitigated by team flexibility and willing communication. ■

References:

Bayer, S. and J. Highsmitt. "Radical Software Development," *American Programmer*, June 1994.

Dixon, R. *Winning with CASE*. New York: McGraw-Hill, 1992.

Hofman, J. D. and J.F. Rockart. "Application Templates: Faster, Better, and Cheaper Systems," *Sloan Management Review*, Fall, 1994.

Jacobson, I. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Reading, Mass.: Addison-Wesley, 1992.

Jones, C. *Assessment and Control of Software Risks*. Englewood Cliffs, N.J., Prentice-Hall 1994.

Kerr, J. M. and R. Hunter. *Inside RAD*. New York City: McGraw-Hill, 1994.

Martin, James. *Rapid Application Development*. New York City: Macmillan, 1991.

Zahniser, Rich. "Timeboxing for Top Team Performance," *Software Development*, March, 1994.