Rapid Delivery: An evolutionary approach for application development

by D. Hough

From a historical vantage point, large application development projects are frequently at risk of failure. Applications are typically developed using a monolithic development approach. Monolithic approaches generally feature business-user-defined requirements that are incorporated in the application but not evident until the resulting application has been implemented. To effectively produce new information systems, innovative methods must be utilized. This paper provides information about one of these, Rapid Delivery—a method for developing applications that can evolve over time. To fully understand the principles of Rapid Delivery, a discussion is included that illuminates a three-dimensional application model and its variations. The application model helps in understanding application segmentation, a technique used in Rapid Delivery to break applications into a variety of functional capabilities. After the development of each application segment has been completed, it is implemented to provide immediate benefit to the enterprise; each application segment is added to the evolving application and its ever-expanding capabilities. The result of using Rapid Delivery is an enhanced ability to build applications that better support the enterprise through a continuous stream of delivered requirements, a reduction in the possibility of project failure, and a diminished likelihood of runaway projects.

Time is an element that can be used to our advantage but frequently seems to work against us. In application development, time has traditionally been one of the foremost problems causing risk to escalate, acting as a restrictor of what can be accomplished within allotted development periods and reducing or eliminating the competitive edge from an application, product, or service.

To manage time to our advantage for application development, we must change the generally prevalent philosophy. In the past, this philosophy has framed application development as a process that follows a long, unbending path, resulting in software that is anything but "soft." Many applications are developed using a monolithic approach even though business needs and requirements change over time. D. R. Graham defines a monolithic development approach as one that regards "development as one large process to be considered in its entirety." The waterfall life-cycle model is one example of a monolithic approach for application development. Instances are common where monolithic approaches have been used in which two-year application development projects have taken six years, two-and-one-half-year projects have had repeated project scoping changes, and five-year projects have never been completed.[1]

To begin changing the development philosophy, or the mind-set really, we discuss application development as a process that leverages time advantageously. By leveraging time to our advantage, we can realize applications that can be developed in a responsive manner: responsive to changes in business, responsive to user needs, and responsive to changes that cannot be forecasted. This concept also supports common business practices in which premiums are charged for products that can be delivered within advantageous time frames.

Delivering applications that can evolve over time is a second key element in the philosophy change. Past experiences suggest that attempts at building large-scale applications have had a less-than-desirable outcome when evolution or change is not considered. This negative outcome is caused by trying to use a monolithic development approach which, in turn, results in applications that no longer suit the needs of the business. This philosophy can be realized by developing and delivering portions of applications rather than developing the total application before it is delivered. This evolutionary process reduces the complexity of each application portion and provides application functions over time to consumers, customers, and clients. An important facet of the evolutionary approach is that it can reduce overall I/S (information systems) development, support, and maintenance costs by anticipating, or perhaps almost expecting, changes to the application. As changes in the application are accommodated, overall maintenance costs for the application should decrease dramatically.

It follows then, that if applications are developed over time (and thus evolve), they can address the needs of the business as it changes, downsizes, and adopts new technologies. Business climates change so rapidly that some corporations report the life expectancy of certain product lines as being only three months.[2] The contribution of an application to the bottom line of an enterprise is being scrutinized more closely than ever. By having applications evolve, the business needs of each corporation can be met.

We describe Rapid Delivery in this paper, a name used to identify a development approach that has been widely used by many software development companies. Rapid Delivery has several similarities to a Japanese approach to product evolution called *product churning*.[3,4] In this paper, we define terms used throughout the paper and discuss the primary concepts of Rapid Delivery. A dimensional application model and its variations are described and are used to assist the reader in understanding the foundational concepts and principles. An overview of the method is presented with a discussion about the need for a firm application architecture to accommodate this approach. Finally, the activities involved in Rapid Delivery are described.

The problems of large development projects

One of the major flaws of monolithic application development approaches is the fact that all projects are considered as being equivalent, that is, equivalent in aspects such as application type, complexity, and time to develop. The *duration effect* is a phenomenon that is exhibited in projects that persist for too long, thereby impacting the manageability and outcome of the project. In a paper written in the early 1980s, Paul Melichar[5] wrote about applications that suffer from the duration effect. In summary, Melichar described the symptoms as manifesting themselves in application development projects in the following ways:

- A slowed pace of development
- Treatment of the project as a "career"
- Lost sight of the original business problem
- Loss of interest by the development staff
- Constantly changing user requirements
- Deterioration of morale in individuals participating in the development project
- Intense, often futile attempts to speed up the development of the application, often resulting in confusion and reduced project manageability

In addition, large application development projects, including those that take a long time to develop, suffer a proportionally large risk of failure, where the project:

- Was implemented but was unequivocally inadequate to the end user(s)
- Was implemented but had a negative impact on the corporation
- Was abandoned because of business or technical analysis that concluded the application would prove to be a failure if implemented

Typically, applications are viewed as a whole. If the project is large, the associated risk is deemed

to be proportionately large. Smaller projects generally are associated with a lower risk. When application development risk is considered, the following points are of consequence:

- Correctly captured user requirements
- Adequate and correct attainment of project estimates
- Appropriately monitored project management throughout the life of the development effort
- Checks and balances that help project participants keep project objectives clearly in mind

There are other elements affecting the outcome of long-term application development projects. Often, progress is difficult to communicate to the users, especially in large-scale projects. In large projects, this poor user perception and morale may transfer to the developers. Once the application has been developed, using traditional development approaches, maintenance of the application ensues immediately after the application is delivered. Many of these maintenance "enhancements" result from user needs that have changed after the initial requirements were gathered or from requirements that were missed.

Another aspect of large-scale application development projects is the notion of the "runaway" project. Such projects, according to the study[6] of one accounting firm, comprise 35 percent of the total projects of the 600 largest clients of the firm. Rothfeder defines a runaway project as "a system that is millions [of dollars] over budget, years behind schedule, and—if ever completed—less effective than promised."[7] Poor planning often causes these projects to run away unchecked since a large project has many tasks, each of which must be adequately addressed; to plan otherwise can and will jeopardize entire projects. There are many cases in which long-duration projects have been compressed from a several-year period to a one-year period. This project compression generally is approached by adding additional resources, making the project even more difficult to properly manage. Project compression can also be required because of existing, defined tasks that must be expedited as their completion is falling behind schedule. Sometimes unattainable deadlines are set, causing individuals to work in a frenzy rather than setting the expectation that regardless of the deadline, the work simply cannot be accomplished within the specified time frames. An additional aspect of runaway projects comes in the form of changes, usually many of them that are outside the scope of the original project. It is conceivable then, that by the time a large-scale system has been implemented, changes that come after the implementation can require a number of other changes that outweigh the original total development effort.

The development of software in chunks is not a new concept. IBM has developed products using the release or versioning concept for a number of years, as have other providers of software. This method allows software providers the flexibility to develop all or part of a product to fit within a "window" of opportunity—that is, to be responsive to the markets they serve. It is important to know, however, that Rapid Delivery is not intended to support this specific condition, rather it is intended to help in meeting other business pressures, such as those related to competition, quality, and cost.

A final point related to Rapid Delivery focuses on having users accept parts of applications. Traditionally, the I/S function has failed to ask users if a portion of an application would be of benefit. Many times this would be true if the alternatives were positioned as a choice: would the users like nothing until two years of work have been completed, or would subsets of the application be of value, being delivered in shorter, regular time periods.

Definitions and concepts

Applications can be viewed as having *functional capabilities*. A functional capability is a grouping of application requirements that, once developed, results in a faculty of an application such as querying an airline reservation, for example. A logical collection of one or more functional capabilities is called an *application segment.*

Rapid Delivery is a building-block approach to large-scale system development projects. Projects are partitioned so that functional capabilities are provided to the users on a regular basis (e.g., every six to eighteen months, or even less). As each application segment is completed, the parts of the application thus available are put into production. By delivering portions of the application, the project becomes easier to manage. The success rate for large projects is notably improved by controlling the term of a project through segmen-

tation into several, more easily managed projects which, when combined, equal the single, large project. Rapid Delivery does not mean that one application segment must be completed before another can be started—application segments can be developed in an overlapping manner.

The prime incentive for Rapid Delivery is the very high risk of failure associated with projects that last longer than two years. Rapid Delivery helps to reduce the effects brought on by project risk, discussed earlier. Rapid Delivery can also be used to have the majority of applications, other than those deemed trivial, evolve to their desired state over time. The fact that application segments must be highly independent of one another to be integrated, implemented, and tested is a key to Rapid Delivery. A major advantage of Rapid Delivery is that it delivers function to users much earlier than traditional development methods with associated benefits to the enterprise and gives developers feedback as development progresses. Studies have shown that even "A small increment of the system functionality, even five percent, was of use to the sponsor ..."[8]

Rapid Delivery can help to:

1. Establish an information system basis that can evolve over time
2. Enable the implementation of portions of applications in a systematic manner
3. Accommodate changing business needs
4. Provide adaptability in the way the application is to be developed
5. Deliver earlier benefits to users
6. Reduce the risks of runaway projects
7. Assist in gaining user confidence early in the process

The application dimensional model

Developed applications have several dimensions as shown in Figure 1. When determining the requirements and design specifications for an application, the width, height, and depth of each dimension is defined. We illustrate this concept graphically because it is difficult to describe an abstract concept in words. The dimensional model is a convenient basis for discussions where the conveyance of key concepts is critical; it can be easily explained and understood by a variety of individuals, whether their orientation is technical or nontechnical. Throughout this paper, we refer to the dimensional model and its variations.

The coloring seen in succeeding figures is used for purposes of delineation between planes of the application dimensional model. Each dimension of the model is described below.

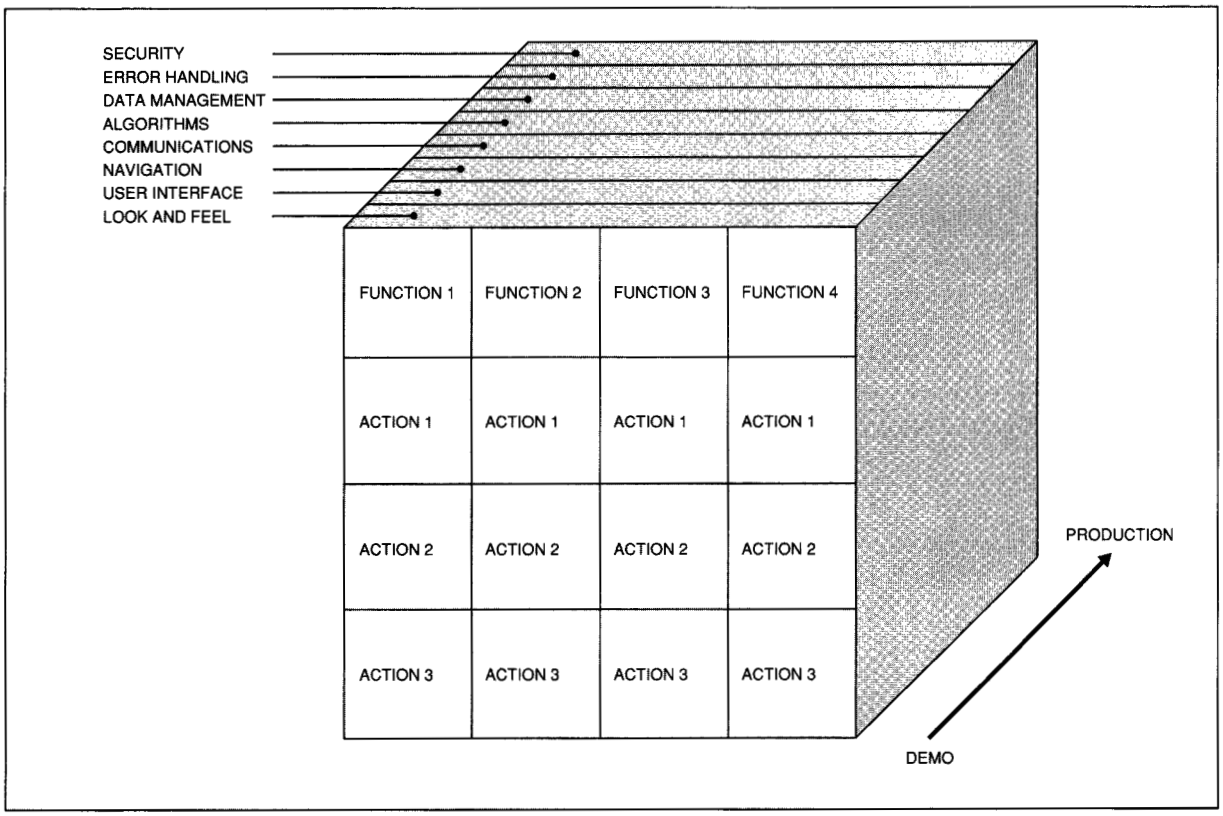
The violet-shaded, top row, side-to-side width on the face of the dimensional model represents *functions* or *objects*.[9] For example, Function 1 could symbolize the function "process notes." In a different vein, Function 2 could represent the object "file." Vertically, beneath each function or object, are the *actions* associated with each function or object. The actions associated with Function 1 might be "send a note" (Action 1), "view the note log" (Action 2), "change the note log" (Action 3), and "print the note log" (Action 4). For Function 2 (object) "file," the associated actions (Action 1 through Action 5) might include "new," "open," "save," "save as," and "delete." Note that functions (objects) can have varying numbers of associated actions depending on each function and the supporting requirements.

The depth of the dimensional model defines the attributes, or associated application characteristics that increase the substance of the application. The shallower this depth is defined, the less robust is the application. "Look and feel," "user interface," and "navigation" are examples of these elements of refinement. As the depth increases, the software becomes increasingly like a production system. Applications have varying depths, depending on the specific application characteristics and to what degree these attributes are defined. For example, certain applications require more depth for "error handling" than for "security." The terms "bullet-proof" or "industrial-strength" are sometimes used to describe production-level code. The associated application characteristics indicated on the top of the model as shown in Figure 1 are typical of production systems, although the list is not all-inclusive.

Application model variations

The total model (e.g., all of the dimensions) illustrated in Figure 1 conceptually represents a complete, production-level application. Any portion less than the whole of the dimensional model is a representation of a subset of the final application.

Figure 1 Application dimensional model



For each application, the application dimensional model appears differently. The number of variations is practically limitless. Representative variations will be discussed in this section to show the concept of variations on the dimensional model.

The model in Figure 2 shows an application segment with little depth developed; basically, a facade is built. This model could be used to describe the end-user interface (EUI) components of an application. Because of its shallowness, it may be useful for communicating the degree of development necessary for demonstrating screens, action bars, and windows. Application segments developed to only this level may be prone to frequent failure since more emphasis and work has been put into the EUI and the overall look and feel of the application than into the development and testing of code that supports features such as error-handling, data management, and validation. This model variation is typical of many application prototypes. It is beneficial to leverage the use of specialized development methods and techniques along with development tools that are designed for iterative construction of the user interface.

The "T-shaped" model illustrated in Figure 3 shows a variation on the application dimensional model that represents an application segment with a reduced user interface and a narrowly defined set of actions associated with a specific application function. This vertical development approach may be appropriate for applications in which functional capabilities can easily be grouped together or are of specific interest. "Slicing" of the application vertically also assists when bounding the work to be performed for a specific application development effort. This model provides an excellent example of one type of an ap-

Figure 2 Facade model variation

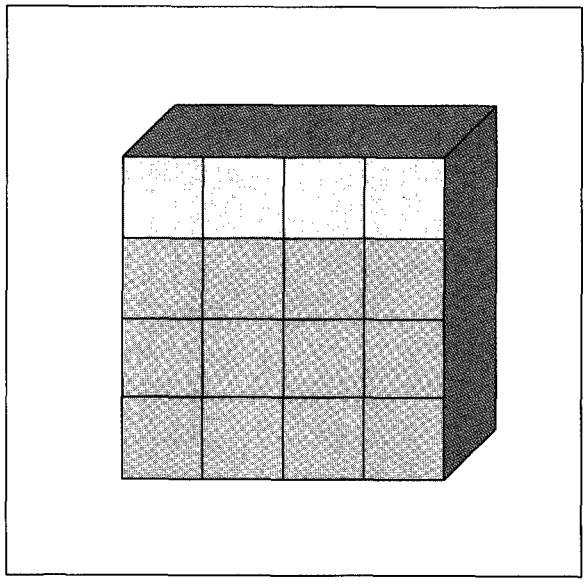
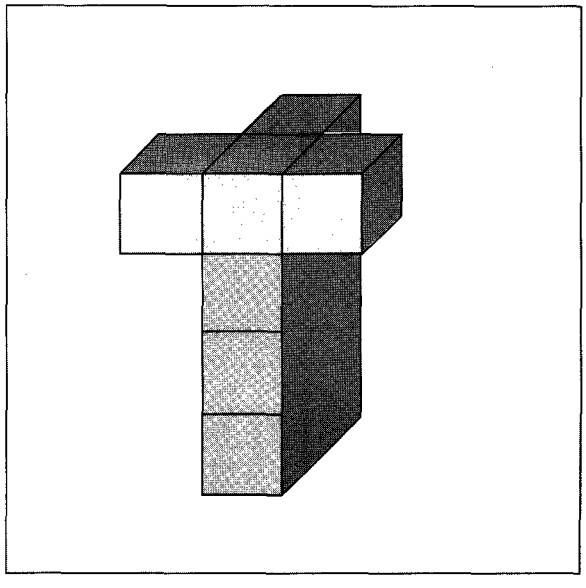


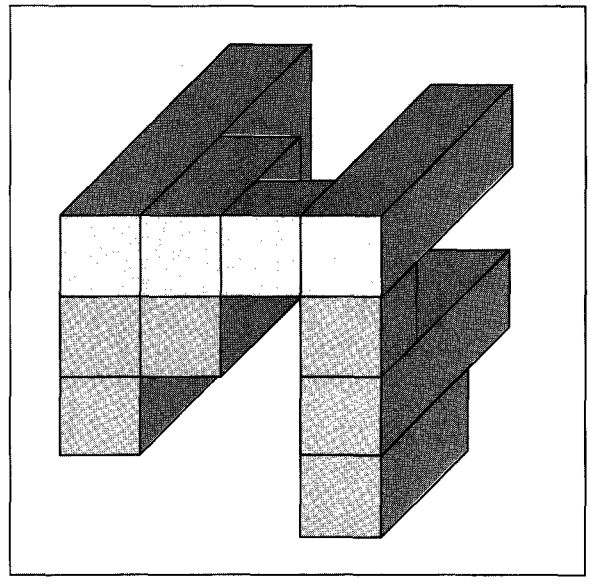
Figure 3 "T"-shaped model variation



plication segment selected for Rapid Delivery, illustrating what is to be developed and delivered.

Figure 4 depicts a model in which several functional capabilities of the application have been

Figure 4 Unevenly shaped model variation

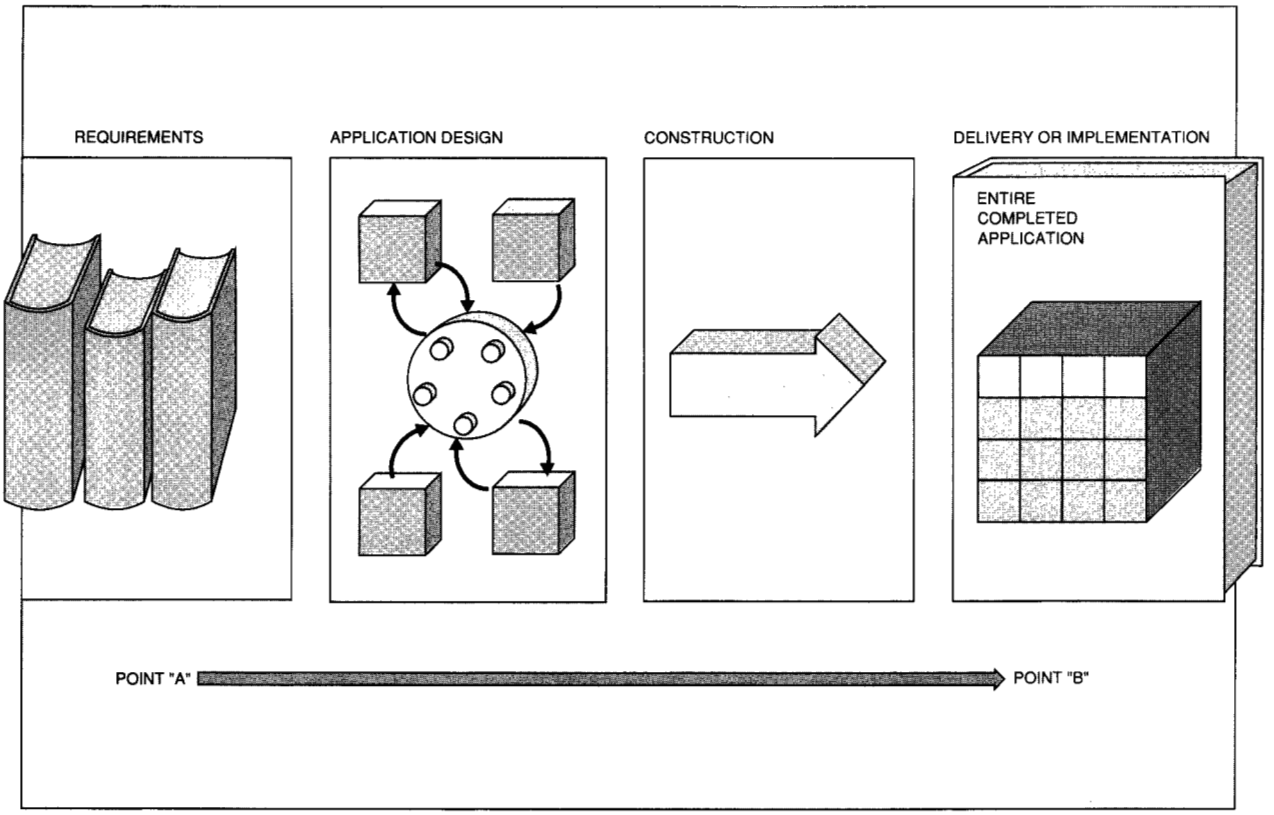


defined, each to a different depth. Additionally, there is unevenness in the number of defined actions associated with each function. Notice too the uneven depth shown from the front to the back of the illustration, down to the level of the associated actions of a specific function. The unevenness of this model describes an application segment that will be developed with varying degrees of functionality. Some actions may be demonstrated with little functionality in place, whereas other actions might require more extensive development to provide a fully operational application segment.

The importance of an architectural foundation

To realize the full potential of an application, it must be based on a sound architecture. This means that design of the application alone is not sufficient. It is relatively easy to design a simple house (e.g., one with three bedrooms, a kitchen, etc.); however, only individuals with specific skills can properly design the architecture of a 50-story building (how the walls are built, how big the foundation must be, etc.). Design implies caring for today's needs, whereas architecture suggests that long-term issues, including expansion, have been taken into consideration. Therefore, of

Figure 5 Traditional application development



key importance to Rapid Delivery is the ability to continuously add increasing functionality to the evolving application. If the overall application architecture is not sufficiently addressed early on, it will be difficult to integrate application segments over time. A few architectures to consider, in addition to application, data, and process architectures are: presentation, control, security, and communications architectures.

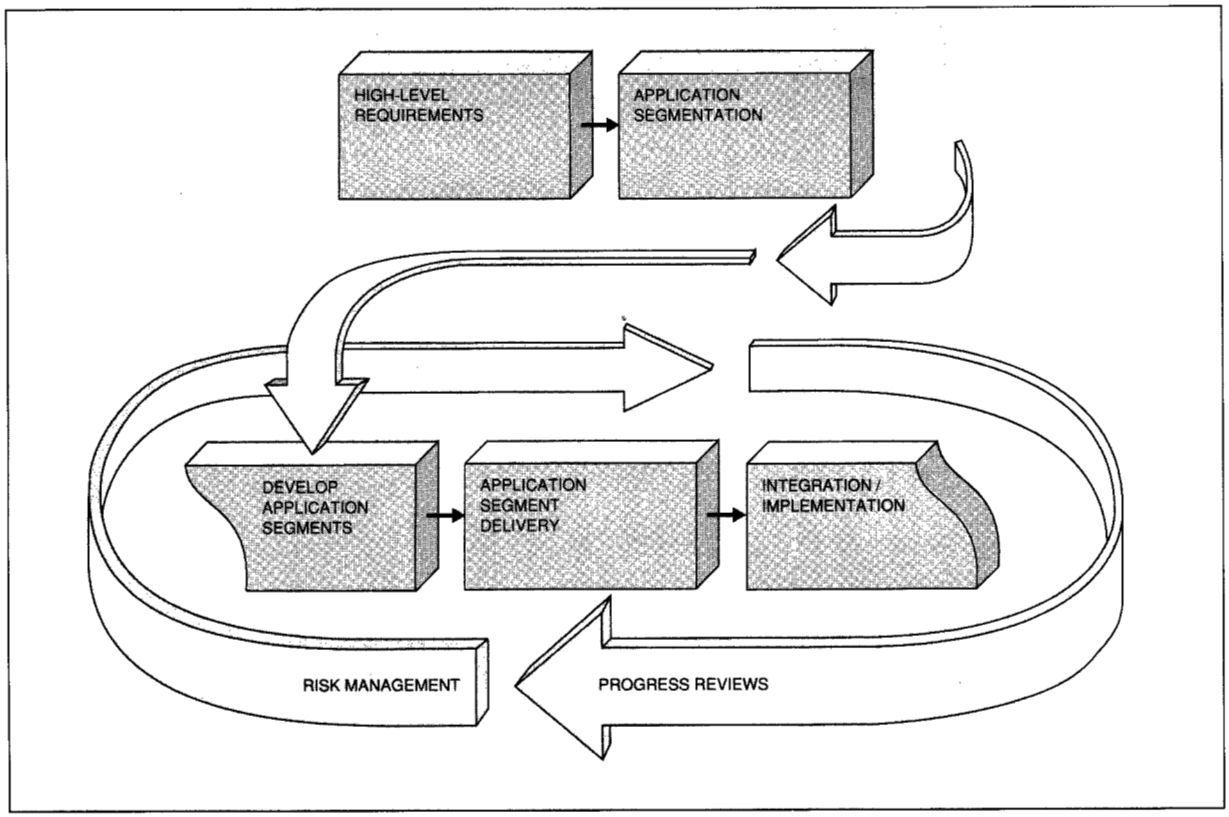
Having an experienced application architect is more important to Rapid Delivery projects than to other types of application development projects. It is particularly important because application segments must be constructed so that each fits with other application segments in a cohesive way. If the application architecture is in question, an option is to build a proof-of-concept prototype that will allow the development team to determine whether its intended architecture and application segment design can coexist in a quality manner.

Having an application architecture also yields a better understanding of the overall application. If, during application construction, the application becomes too large or unwieldy, the architecture should allow the application architect to divide the application into additional application segments. Segmentation makes the overall application simpler and less complex to develop, test, and implement.

Overview of the method

Rapid Delivery, as an approach, develops parts of an application using all of the methods, techniques, and processes that comprise traditional development; the development of each part is done as if it were a complete project itself. For comparison, a graphical depiction of "traditional" application development is shown in Figure 5. This illustration is not intended to imply any par-

Figure 6 Rapid Delivery activities



ticular methodological approach, such as information engineering or structured methodologies.

The illustration shows how, in traditional application development, *all* of the application requirements are gathered at the beginning of the project ("Point A"). This activity alone could sometimes take two years or more for large-scale application development projects. After the requirements gathering activity come analysis and design activities. Once the design has been completed for the total application, the application is constructed in its entirety. When the application is totally finished, it is delivered and implemented ("Point B").

Rapid Delivery changes the process for such large, long-duration application development projects; iteration is a key element as well as ensuring user involvement throughout the development process. The same development fundamentals apply; the same development methodologies, the same planning, application requirements, analysis and design, construction, and production and maintenance activities are incorporated in each Rapid Delivery effort. No steps are left out, and no shortcuts are taken. Rapid Delivery has the same need for an enterprise process and data model to be in place as do all application development projects. Note that Rapid Delivery can support the development and delivery of individual application functions faster; however, this can be enabled only after a base of resources and personnel has been accumulated from experience. Rapid Delivery inherently provides the ability to break up and manage projects more effectively and to deliver working functions to the user much earlier.

Figure 6 illustrates the activities that comprise Rapid Delivery. We will examine each activity briefly, then discuss each one in more detail later.

Rapid Delivery activities

First, *high-level requirements* are gathered for the overall application. Requirements are gathered at a level sufficient to set the direction for development of the application. In the *application segmentation* activity, the high-level application requirements are analyzed, and an application segmentation strategy is developed. Once the segmentation strategy is developed and approved, the *develop application segments* activity begins, which is comprised of additional, detailed requirements gathering activities directed toward the specific application segment to be developed using analysis and design, produce, build and test, and production and maintenance activities. These activities are performed in an iterative fashion to ensure that each application segment accurately matches the user requirements. In the *application segment delivery* activity, the completed application segment is delivered and prepared for the succeeding activity. When each application segment is delivered, each segment is integrated with already existing application components and is implemented in a production environment in the *integration/implementation* activity. Notice in Figure 6 how the high-level requirements and application segmentation activities take place outside of the remainder of the activities. *Risk management* and *progress reviews* occur at the beginning of the development process for each application segment and every three months after Rapid Delivery projects start. These ongoing activities are directed toward keeping each Rapid Delivery project on target and to communicate overall project status to date. If redirection is necessary, risk management and progress reviews can be used to determine the corrective actions necessary and to develop a plan for bringing the project within acceptable tolerances.

High-level requirements

The high-level requirements activity is intended to help provide a continuously growing, useful system that can be developed and implemented in a timely manner, providing early benefits to the corporation and end users. While one application segment is being implemented, another segment is being built. It is possible that segments being completed may require changes to the functional capabilities of segments already developed. The potential for rework should be recognized, because requirements and design of later parts may impact the application as it was initially defined. The risk of rework is much lower than the risk of failure of projects lasting more than two years. This risk of rework then becomes a *reasonable* risk.

Organizations that use structured methods typically use a method such as Business System Planning/Architecture (BSP/A) to identify applications or application groups resulting from the process architecture developed during the BSP/A. In contrast, organizations that use information engineering methods commonly use a Business Area Analysis (BAA) method to identify business systems from the process-data affinity analysis developed during the BAA. From either of these methods, the resulting outputs should be used as input to the high-level requirements activity.

The underlying activities in high-level requirements are the following:

1. Define high-level requirements
2. Determine the "outer boundaries" of the application
3. Design the needed high-level functions

Define high-level requirements. Using the Joint Application Design (JAD) method, key enterprise management and users are assembled to discuss the application at hand. During these JAD sessions, the objective is to look at the application from a high level, focusing specifically on functions and requirements that can be easily identified first, followed by other functions and requirements that may not be so obvious. JAD participants should be told that these sessions will not cast the "outer boundaries" of the application in concrete; the JAD sessions are intended to allow the participants to describe the functions and capabilities that must be present in the application as best they can at this point. Specific details should be deferred until those details become important to the function being addressed.

The intention of defining these requirements is to stay at a high-enough level to provide not only for the addition of requirements at a later point in time but also to be detailed enough for the functions and requirements to act as a basis for segmenting the application.

Determine the "outer boundaries" of the application. The next step in Rapid Delivery is a process

of determining the "outer boundaries" of the application. In this sense, outer boundaries are defined as the largest possible dimensions that will contain the application. This process is sometimes called the *forecasting horizon*.[10] Land describes the forecasting horizon as ". . . some time in the future, the uncertainty becomes so great that the systems designers cannot conceive of any design that can cope with the possible range of requirements at a permissible cost." This description suggests that reasonable boundaries should be drawn around the application for development. It should be understood that these application boundaries are intended to be used as a guide or road map to future development of additional application segments. Note that the outer boundaries in real-world business change dynamically; therefore, the boundaries should be considered as the most flexible part of the application. Changes in business strategies or objectives should drive changes to these outer boundaries.

Once the application requirements are identified, a process model should be constructed that relates processes to the identified application functions. This model helps in sequencing the identified application functions for development, the next step in defining the application boundaries. Once the functions have been sequenced, the JAD participants should review the functions to ensure that they have been sequenced for development in an appropriate way.

Design the needed high-level functions. Functional decomposition provides a technique for documenting the high-level design of the application. Each function in the identified sequence should be examined. Each function and more specific requirements for that function should be considered. The question should be asked, "What must this function do?" These requirements should be listed item by item, again at a high level. Precise design specifications, technical issues, and concerns should be deferred until the function is to actually be developed. When it appears that the participants cannot avoid low-level specifications, it is time to conclude the functional decomposition process.

At this point the JAD participants should assess whether enough information exists to logically group the identified application functions and requirements. If insufficient details exist, additional JAD sessions should be conducted to further refine and improve the functions and the listed requirements. This high-level identification of functions and requirements should continue until sufficient information exists to allow the application segmentation activity to occur.

Application segmentation

The objectives of application segmentation are to define application segments that can execute useful functions and to determine a sequence that allows the evolving application to be used in the order delivered. Once the segments and sequences are identified, they should be verified with the users. One of the benefits in segmenting the application into more pieces with shorter deadlines is that the development staff can be used to its capacity.[11] The primary dilemma associated with application segmentation is: how does one divide (segment) the architecture and design of an application?

In the following sections, we look at a number of alternatives. The primary tasks involved in application segmentation are:

1. Develop segmentation strategy
2. Define the sequence for segments to be developed and delivered

Develop segmentation strategy. There are many issues to consider when dividing the application into logical segments. This section looks at the key issues associated with application segmentation.

Once the high-level application functions have been identified, they should be grouped together. This logical grouping should place related application functions together (e.g., user interface application elements). It is recommended that the "front" of the application be built first (e.g., the user interface). The segments should be small enough to reduce the risk involved with changing requirements and to ensure that the segments can be built in a time frame of six to eighteen months (or less). Less than six months will typically not be enough time to adequately develop a complicated segment. It is better to have numerous segments with a low level of technical complexity than a small number of segments with a high degree of technical complexity. When obvious and overwhelming complexities exist, consider breaking these segments into simpler pieces to

make their development less difficult. The segmentation also helps to boost overall morale as parts of the application are constantly being delivered, and it provides the users with new functions to review. Application requirements that are not well understood or are "fuzzy" should be prototyped.

The key characteristic that must be maintained in order for Rapid Delivery to be successful is that the segments must be highly independent. If the segments are not independent, division into segments can increase rather than decrease the complexity and also increases the chance for a runaway project. Finding application segments that can be delivered and are ready for the users will, in itself, require that the segments be fairly independent. Still, there are many ways to divide applications into independent segments. We discuss these next.

Business strategy. Since business strategies, goals, and objectives are assumed to already be in place, segmentation of the application to support the most important business strategies may prove to be an effective segmentation strategy. In this way, the corporation should realize an impact on these business strategies such as:

- Increasing contribution to the bottom line
- Prospering in competitive advantage situations (e.g., first to market)
- Taking advantage of specific market opportunities
- Delving into business areas resulting in increased market share

Global geography. As more companies grow on a multinational scale, it may be important to develop portions of the application that are generic in nature. Functionality capabilities that are common, for example, to Canada, Japan, and France, can be developed and delivered in a series of application segments. Where unique functions are necessary, they can be developed as additional application segments that can be delivered to specific countries or optionally to all countries, whether those functions are used or not. In developing applications in this manner, cultural and language barriers must be addressed to ensure that initial application segments truly are common across all of the international locations.

Data usage. It is possible to divide the application on the basis of data usage. Data should support one or more business processes. When the data have been identified, the application can be segmented in such a way that each business process and its associated data together provide a functional user capability.

Most typical functions. Another way of segmenting the application is to determine the most commonly used functions. When these are determined, it may still prove to be too much function to place into an application segment. A further breakdown may be required to define more segments that are smaller and less complex technically.

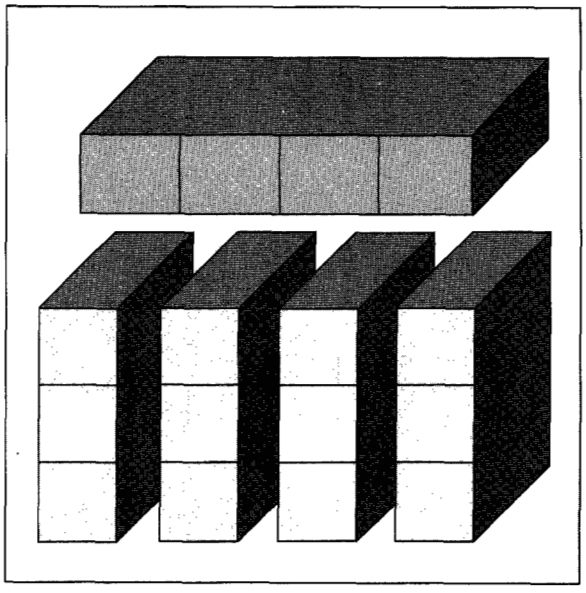
Simplest functions first. It may become important to establish I/S department credibility with the user community early in the project. The perception that progress is being made can have a positive influence on management, users, and the development staff. The same management team that was reluctant to allocate budget dollars to a large-scale project may support projects developed with the Rapid Delivery method. Users and the development staff gain confidence when they see application segments and accompanying capabilities implemented in a timely manner. This means the simplest functions can be developed quicker, resulting in delivering functions to the user faster.

This approach is similar to the strategies used to solve jigsaw puzzles. The simplest way to solve a puzzle is to first assemble like colors or shapes (e.g., the border of the puzzle with straight edges). Once several segments have been put together, it is easier to join unlike pieces to the puzzle chunks that have already been completed until the puzzle is finished.

Most-willing users. With the type of user community in mind, consider segmenting the application by the lines of business, divisions, or departments most willing to accept parts of the application that will be implemented over time. Individuals frequently are interested in certain portions of an application and are ready to commit to the project.

Budgetary considerations. Budgetary considerations are almost always a leading issue. It may be most beneficial financially and even politically to develop application segments that give the most value at the least cost. These situations should be analyzed carefully to ensure that the appropriate

Figure 7 Rapid Delivery horizontal and vertical application segmentation



application segments are developed, particularly if the decision made means that development costs may be higher using Rapid Delivery instead of another development approach.

Organizational segmentation. Segmentation of the application by organizational entity may provide the level of segmentation desired. A particular line of business, or a specific corporate division or department, may benefit from their corresponding application segments being developed first. Remember that objections may arise when one organizational unit is "favored" over another. When applications are segmented by organization, application segments can be implemented in the least disruptive fashion for other departments or organizational units. Note how data will flow from unit to unit within the organization and deliver functions to the "upstream" units first.

Build vertical, horizontal, or combination segments. Consider dividing the application in such a way that segments can be built vertically, horizontally, or in a combination of both. Figure 7 illustrates this concept.

Dividing the application in this manner allows it to be built more easily. In this example, the top layer of the figure might indicate the user interface while the vertical sections might represent discrete functions of the application.

Systems and subsystems. If the application consists of subsystems, segment the application into subsystems. If the subsystems are still too large or complex to be managed easily, the subsystems should be further divided into additional application segments.

Build to fit contiguous fragments of time. By defining fixed blocks of time, application segments can be divided to fit into these time frames. The time frames that may be most useful will typically be from 90 to 120 days in duration. These application segments can then be developed in a contiguous fashion until all of the segments have been built. This strategy should only be used when other strategies cannot be used.

Combined segmentation strategy. It may be appropriate to divide the application by using a combination of the previously discussed strategies. For example, a combined strategy of developing the simplest functions with the most typical functions may provide the most flexibility in segmenting the application for the long term. Consider combinations of segmentation strategies when a single strategy will not provide the application partitioning needed. Note that combining two segmentation strategies many times also incorporates other segmentation strategies directly or indirectly. For example, by segmenting the application by building horizontal and vertical segments combined with developing the simplest functions first, budgetary considerations may be directly supported as well. In this example, dividing the application in this manner supports a combination of three segmentation strategies.

Sequence the segments. Once the segmentation strategy has been selected, the application should be segmented into its pieces. When the segmenting is complete, the segments should be sequenced for development. As the segment sequence is determined, target implementation dates should be established for each segment. This sequencing will determine how each segment will overlap in the overall development.

At this point, the segmentation, sequencing, development, and implementation dates should be

documented in a project plan for tracking and communication purposes. Note that each project could be developed in a "single-threaded" fashion, with no more than one segment being developed at a time. A preferable approach is to overlap the projects, dividing the developers into small, independent teams. Once the application segmentation and sequencing have been completed, an estimate of the work necessary to build the first application segment should be determined.

Develop application segments

Development of each application segment includes additional requirements gathering, analysis, design, coding, and testing activities. Rapid Delivery should use the same techniques employed on any six-to-eighteen-month project. This section will not discuss in detail the elements of application development such as analysis, design, diagramming techniques, coding, testing, or other development-specific topics since these elements are discussed in numerous books, articles, and papers. Incorporated in these techniques should be iteration, an approach that places the analysis and design and construction activities into a repeated cycle for a prespecified number of iterations. If inordinate numbers of application functions must be pushed into later application segments, it is a sign of trouble indicating that requirements have not been well established or that communications are inadequate.[8]

The requirements, design specifications, functions, and features for each segment should be determined in more detail. Subject matter experts should be brought together where additional JAD sessions can be conducted to determine these requirements. This definition should provide sufficient information to allow detailed analysis and design activities to proceed in the application segment development activity.

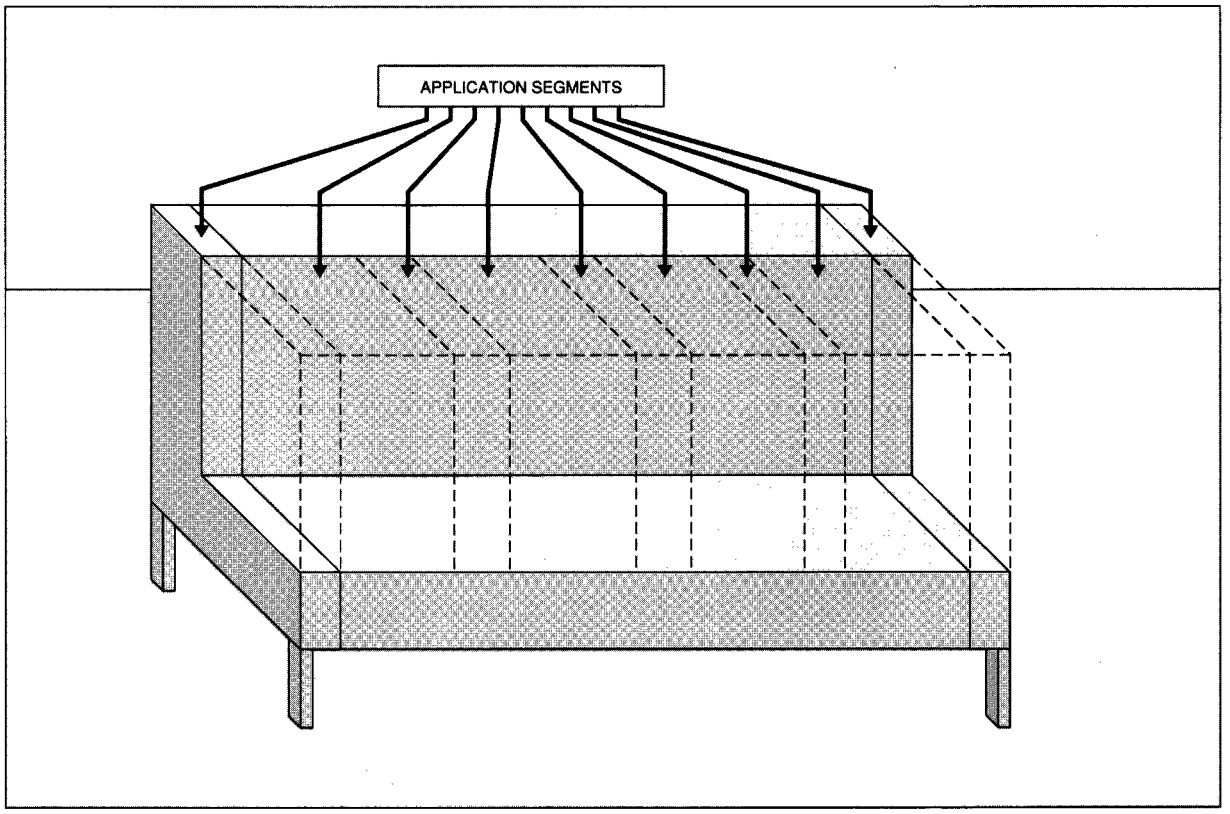
Design for maximum modularity. Provisions should be made within the overall application for the most modularity possible. Applications under development will have portions that will be developed in a later segment, leaving a gap in the software. This gap may need to be filled by "stubbing out" the function or module that will be developed later. Stubbing out refers to application code that is incomplete yet allows the application to be compiled and operate. In designing this modularity, the applications that must be initially stubbed out should be designed with the capability of easily accommodating future segments of the application. This capability is referenced here as a snap-in/snap-out concept. Snap-in/snap-out provides the ability to snap one application segment onto the initial application segment. This snap-on capability should allow modules to be snapped on (or off) from the "top," "bottom," and "sides" of the application dimensional model.

Design for organizational flexibility. Frank Land states that "The extent to which a system responds to changes is not only a function of the way the system is constructed but also the type of organisation in which the system operates. Some forms of organisation are inherently more capable of responding to change than others."[10] Land continues, "In considering the question of lifespan and flexibility, the design has to be aware of any constraints imposed by the style of management and the structure of the organisation." Consider these points when designing parts of the system that may need to be changed because of organizational changes. Also keep these points in mind when changes in the application segment sequence must be evaluated.

Design for extendability—the park bench concept. As already discussed, allowances must be made in the application architecture and design to accept additional segments. One concept, the "park bench," is useful as an approach for the foundational application architecture and design. The park bench is used to establish an overall architectural framework for the application. Once the overall application architecture has been defined, the portion of the architecture that will support the eventual development of the remaining application segments is developed. The initial architectural framework resembles a park bench having a back, a seat, and defined sides as indicated in Figure 8. As each application segment is developed, stubs are removed from the preceding application segment(s), and the subsequent application segments are snapped onto the pre-existing park bench, where the segments are integrated as a part of the total application.

Design to accommodate changing requirements. Categorize application elements according to

Figure 8 Application segments–park bench concept



Adapted from D. R. Graham,"Incremental Development: Review of Nonmonolithic Life-Cycle Development Models," *Information and Software Technology*, Vol. 31, No. 1 (January/February 1989).

whether they are likely to change, they might change, or they most likely will never change. Application elements that are most likely to change should be implemented later, if possible, so that there will be as little impact as possible to the application when the changes actually occur. Application elements that might change should be evaluated for ways in which they can be made less likely to change and what is required from a maintenance standpoint.

Include rigorous configuration management. With Rapid Delivery, configuration management is especially important for separating various application segments under development concurrently. Configuration management is used to identify, organize, and control changes being made to the application (segments) made by the developers. Do not underestimate the effort involved in configuration and versioning of application segments. Configuration management will gain increasing significance as a greater number of application segments have been completed. For applications that are developed on one platform and executed on another, ensure that the configuration management tools can manage across multiple environments. As testing progresses, use configuration management to administer the influx of application defects and necessary application changes.

Ensure that appropriate test management is addressed. The importance of testing within Rapid Delivery projects is no less important than in any other project. It tends to become somewhat more complex to manage because multiple application

segments may be developed concurrently. In fact, Rapid Delivery assists in the gradual enlargement of the test case "base." This enlargement allows the test cases to be developed by different testing specialists for different application segments. Ensure that testing requirements and test cases are addressed at the same time as application requirements. Communication and test case reuse is the key to ensuring that test cases are not specified redundantly. Testing can be enhanced as application segments are integrated, reducing the overall application size and complexity that must be tested.

Manage application changes prudently. Careful management of change requests is of paramount importance with large projects, with or without Rapid Delivery. Changes that affect the general acceptance of the application cannot be ignored. On the other end of the spectrum, numerous small changes can cause grievous damage to project schedules and to project scope. Very frequently, changes are precisely one of the elements causing large-scale development projects to fail. Use rigorous change management methods and techniques to deal with changes in the most effective way.

Application segment delivery, integration, and implementation

Segment delivery is an appropriate place to address expectations management and "project restraint." As each application segment is delivered, the development pace of the next application segment(s), in general, appears to increase. This increase in pace may be due to the following:

- Code reuse—Code reuse should become more prominent as each application segment is developed. As each succeeding application segment is developed, less initial code should have to be developed, since some of the necessary code will already exist.
- Increased expertise—As the development of each application proceeds, the developers' expertise will increase as they become more familiar with the hardware and software used for developing the application. Additionally, the development staff will gain knowledge and confidence using Rapid Delivery.
- Better understanding of the application—As development progresses, users, development staff, and management will increasingly understand the application under development.

Ensure that expectations are set so that individuals are not under the impression that the remaining work will take place faster. Properly setting expectations at this point helps to offset the urgency for developing other application segments by using shortcuts or cutting corners. Cutting corners at this point can lead to failure of the project. Since Rapid Delivery is designed to reduce project risk and to increase the success rate for large-scale projects, it is prudent to exercise "project restraint" and better judgment in moving the development of additional application segments ahead. Since specific plans have been made for the development of the application segments during the application segmentation activity, management and users should not capriciously attempt to change development, and therefore delivery and implementation schedules.

Aristotle said, "What we have to learn to do, we learn by doing." A recommendation, therefore, is to develop a document to capture lessons learned during the segmentation, development, delivery, integration, and implementation activities. This document should provide information describing both the especially effective approaches as well as the unsuccessful approaches to segmentation, delivery, integration, and implementation. By documenting this information, these practices can be exploited or avoided in future efforts.

Finally, be prepared for personnel contention. Users may be involved in more than one Rapid Delivery project concurrently. The development staff may want users' attention focused on several of the Rapid Delivery projects underway all at the same time. Recognizing that this contention will likely take place will allow the scheduling of users and enable the proper individuals to be involved in the appropriate time frames.

Delivery. We define delivery as the completion of one or more operating application segments. These will be realized after the application segment development activity has been completed. Delivery should be performed using similar techniques as those used with "traditional" application development projects. Several elements apply differently to Rapid Delivery and are discussed here.

Delivery dates will inevitably become a focal point in the life cycle of the project. The temptation is to slip delivery dates to accommodate

additional work to make the application more appealing to the user or to increase its acceptance. However, do not slip the delivery date. Generally, when dates are extended, additional work fills the additional time. Productivity decreases and harmful after-effects may enter into an otherwise successful project.[12]

Delivery also implies packaging the application, which means that documentation should be completed for each application segment. This packaging differs from the "traditional" application packaging in that each application segment should be completely documented rather than creating a single document for the entire application. Installation issues should be resolved, prerequisites should be identified, and additional elements that should be a part of this packaging and are specific to each application segment should be finished.

The users should be prepared for the implementation of each application segment and the ensuing availability of the additional application function(s). They should also be prepared to use what they specified. Again under the category of expectations management, the users should be reminded that the application segment being delivered will do exactly what was specified by the user since iteration has been used during the development process.

Integration. Integration "marries" each application segment to those already produced. During the integration, the current application segment is placed on the "park bench." As each application segment is integrated, the segment should be snapped onto the other pre-existing portions of the application as described earlier.

Implementation. What remains is implementation of the application into the production environment. The implementation should be scheduled for immediate cutover since each delivered application segment provides continuity of the application and additional functionality not previously available.

Risk management and progress reviews

Use risk management and progress reviews to regularly monitor any initial and current risk factors involved with the project and to assess the progress of application segment development. Risk management techniques recognize problems as early as possible, deal with the identified risks directly, and reduce the overall risks of the project. Progress reviews assess the total application, the application segments under development, and the application segments to be developed and compare current progress against the plan. Progress reviews yield a picture of the total development effort underway, regardless of the number of simultaneously developed projects and application segments. Figure 9 illustrates the relationship between risk management, progress reviews, and Rapid Delivery application segments.

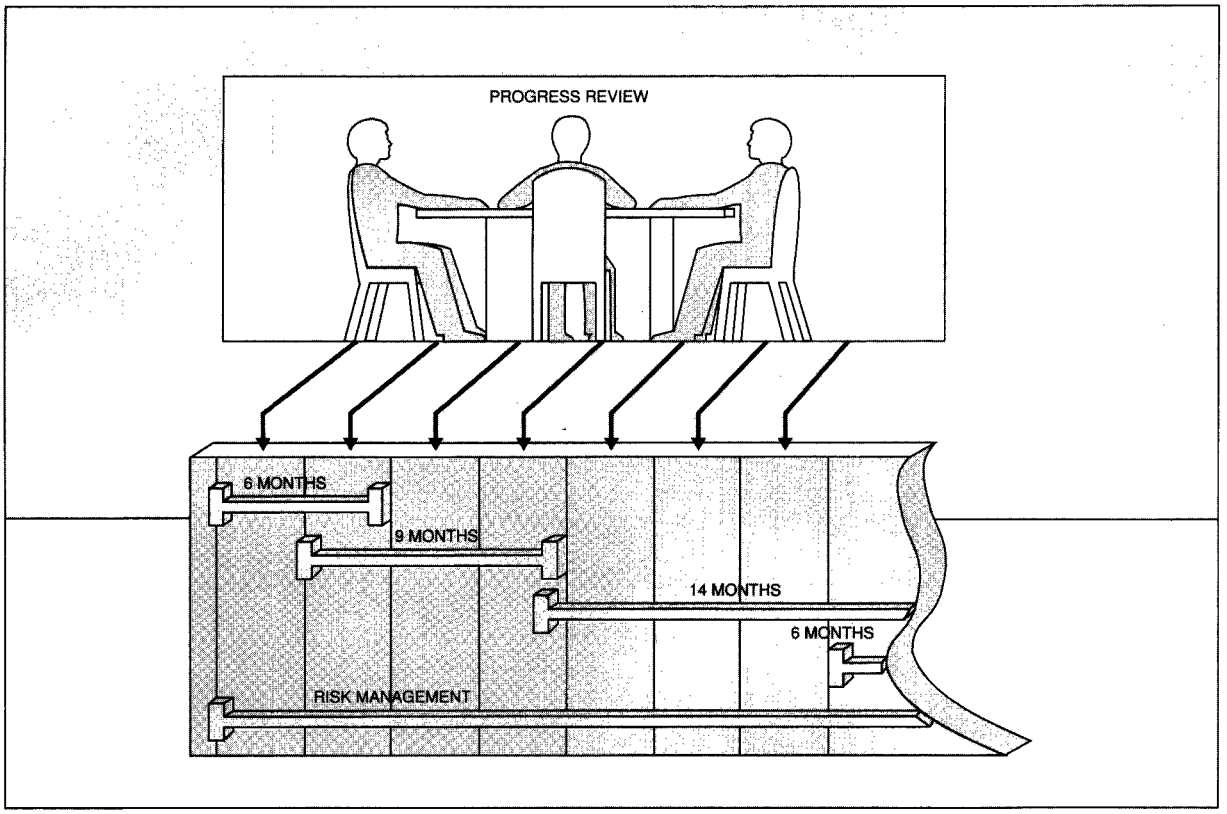
This figure shows a portion of a project plan that indicates the relationship between progress reviews and application segment development. It indicates how progress reviews occur every three months. Risk management is shown at the bottom and extends throughout the life cycle of all ongoing Rapid Delivery projects. By conducting ongoing, scheduled progress reviews, the progress of the application can be assessed.

Risk management. Risk management requires the commitment and participation of corporate executives, I/S management, and development personnel. It monitors Rapid Delivery projects and improves on traditional development approaches by identifying potential risks, recognizing existing risks, and reducing project risks before they jeopardize the large-scale project. These risks have been discovered by others where a ". . . software risk-reduction phase is appropriate, even for a software-based system not requiring hardware development or special integration."[12] Risk management evaluates each application segment being developed and provides a standard against which the ongoing risk management process can be measured. The outcome of the risk management activity may lead to a determination of whether a project should be continued, adjusted, or terminated.

Risk management should examine questions unique to Rapid Delivery such as the following:

- If development problems exist, should they be corrected before going on with the development of the remaining application segments?
- Should development be stopped until existing problems are fixed before developing the next application segment(s)?

Figure 9 Relationship of risk management, progress reviews, and Rapid Delivery application segments



- Should development efforts simply continue while simultaneously trying to resolve existing problems?

At the inception of each Rapid Delivery project, an initial risk assessment should be conducted to identify elements of the project that may increase the risk or impact the success of the project. Based on studies performed by McComb and Smith,[6] the following elements are project components that should be initially assessed and then monitored throughout the project:

Planning and executing. The planning and execution factors that must be dealt with carefully are:

- Estimating—Estimates should be evaluated for accuracy. Approaches to estimating are not at issue as much as the accuracy that an approach yields. Ensure that estimates and appropriate personnel are available and that sufficient time has been allotted to complete tasks within realistic time frames.
- Compression—If the project schedule is being compressed, ensure that the schedule is realistic. The schedule should be one that can be accomplished without resorting to unusual steps to attain it.
- Change management—Bad change management can cause a large project to fail, almost overnight. Assure the development staff that changes will be managed effectively and that changes affecting development activities will be properly processed. This assurance alone should greatly enhance the development process and should reduce the frustrations brought about by a steady stream of change requests. Changes that have been mismanaged or not managed at all can certainly cause a project to fail.
- Workarounds—Ensure that problems are dealt with directly rather than being "creatively" re-

solved. Circumventing certain business practices such as appropriate use of naming conventions and standards inevitably causes problems. Make sure that inconsistencies such as work estimates, resources, and delivery dates are identified and properly resolved. Pinpoint dependencies and make sure that dependent project elements are dealt with appropriately. Elements such as hardware that must be in place before another task can begin is an example of a potential workaround situation. Oftentimes such a situation is managed by performing the testing on existing hardware which may, in turn, impact other work being currently executed on the project.

Human. These project aspects have to do specifically with human factors.

- Bid strategy—When using external contracting resources, be aware that the lowest-priced contractor may also carry the most risk. Low bidders many times have less expertise and, therefore, reduced overall productivity. If work estimates are based on a higher skill base, risk will be introduced into the project if the estimates do not also take skills into account. Alternatively, the highest priced contracting resources should be assessed for the appropriate credentials to ensure the added price is warranted.
- Staffing—Adequate personnel for the project means not only having the proper expertise but also having project personnel who are motivated. When skills are lacking, consider either additional training or soliciting assistance from other more skilled internal or external resources who can bring skills to bear on the current project. Staff turnover can be handled by ensuring that the application segments are defined, when possible, so that the ability to move personnel in and out of the project in an orderly fashion exists.
- Scheduling—Scheduling elements that impact risk and should be assessed are elements such as activity sequencing and scheduling. These activities should support the goals of the project. Assess whether the schedule has some degree of flexibility built into it to allow for the addition or insertion of other activities or projects. Examine the overall sequencing of application segment development to make sure that the sequence of development still applies.
- Feedback—Feedback should be sought throughout the project. Feedback is an important tool that can be used to judge perceived as well as real elements of risk. Information gained by feedback should come from management, users, and the development staff. The feedback received can act as a catalyst in identifying potential risks, recognizing risks that already exist, and indicating the considerations for corrective actions.
- Motivation—Motivational project risks involve project personnel that may be a potential danger to the health of the project. Personnel treating the project more as a career than a short-lived project should be identified. Motivation within the project (or lack of it) can have a noticeable impact on the project. Rewards may handle the motivation aspects of a large-scale project.
- User involvement—Having the appropriate users involved in the project can enhance the project by providing the expertise and knowledge necessary to examine portions of the application. Ensure that users are appropriately trained to use the application that is being produced. Involvement in the development of the application as well as training will ensure that the application is accepted by the user community.

Technical. The technical risks of a project are varied and may be the very elements that most endanger the project. Several of these are discussed below:

- Experimenting with new technology—Of all of the technical risks, experimentation with new technology may carry the highest risk. When technology that has either just been announced or is in a beta (test) form is used in a project, the technology should be assessed to determine whether it is the best technical solution, not just the most current. The amount of available expertise should also be evaluated as well as the availability of support for the technology.
- Technical architecture—Technical architecture that has never before been developed can also present risks in application development. Ensure that technical architectures are consistent, well-designed, durable foundations on which the application segments can be built.
- Control—Ensure that controls within the application are identified early so that they can be incorporated directly in the application design. Controls added later are frequently not easily added and may cause undesirable side effects such as degraded performance.
- Performance—Identify potential performance problems and address them directly. Addition-

ally, locate organizational units that require a specific performance level for the application. Ensure that the application to be built is developed on a platform capable of providing the desired performance.

- Products—Risks common to products used in the project relate to hardware and software. Assess the background of these products carefully to identify products that are not well-known, products that do not have a history of reliability, or products with a poor record of support. Newly introduced products should be closely examined to ensure their suitability to the project.

It is recommended that an initial risk management review be conducted before the start of development for the first application segment. It is also recommended that a risk management review be conducted on a quarterly basis. When conducting a review, assess whether the project is on track or if it is in trouble. If the project is having difficulties caused by a negative risk, corrective actions should be identified and executed. Consider conducting a risk management review on a monthly basis until the project is back on track and risks have been made acceptable.

Progress reviews. Progress reviews are intended to evaluate the overall long-duration project as well as each application segment under development. Progress reviews are directed toward dissemination of information related to where each application segment development is in relation to the overall project. This direction provides the ability to communicate information about where the development work is presently situated and how much work remains until the application is done.

Progress reviews should assess work to date, what is on time, what is not and reasons why not, and the corrective actions required to bring the project into agreeable tolerances. Progress reviews also provide the ability to evaluate new technologies to determine their impact if they are added to the development "recipe." If new technologies are identified, attempt to change the uncertainty of the new technology to a risk. Finally, progress reviews can ensure that the next application segment to be developed is fully supported. Full support means that funding has been or will be approved and that the organization as a whole finds value in the continuation of the Rapid Delivery project. Plan on conducting a progress review every three months.

The most successful Rapid Delivery projects will be the ones where the application segment is developed in the allotted time. Delaying work from one application segment to the next may indicate trouble and frequently means that the application segment was planned too aggressively or too much work was placed into a specific application segment.

The advantages of Rapid Delivery

Rapid Delivery provides the ability to make application capabilities available to the business user in a scheduled, regular fashion. Project abandonment is reduced since risk management and progress reviews are embedded within the method. These activities ensure that risks are identified that otherwise might jeopardize the project. By gathering the requirements for each application segment at the time that they are to be developed, urgent user requirements can be included that otherwise might be excluded or go unrecognized using a monolithic development approach. With this method the application can evolve and respond to changes in business. Since each application segment is developed separately, the risk of underestimating development effort is reduced as estimating is performed for each application segment. These advantages and others are discussed next in further detail.

Provides early contributions to the corporation. Since applications are composed of useful user functional capabilities, each delivery can contribute immediately to corporate profit streams. Similarly, application segments that include competitive advantage components can provide this advantage to the enterprise when the competitive opportunities exist, not later when the advantage can no longer be exploited. Finally, the application segments can be used immediately after they have been made available.

Improvements in behavioral factors. Behavioral factors include mistakes made in estimating project time, budget, staffing, and scheduling. All of these estimates become much easier for projects of six to eighteen months or less than for multiyear projects. Errors are discovered much sooner because each project is shorter. User involvement is critical to the building of systems that

users find to be effective. Beyond these factors are technical considerations. One technical factor that frequently leads to failure is the inclusion of new or "emerging" technologies. Long-duration projects certainly have enough risk without introducing technological factors. Second, technical considerations are often overlooked when assessing an application to be developed—applications may be much more complex than first thought. Rapid Delivery deals directly with these risk elements by incorporating risk management into the development cycle.

As components of Rapid Delivery, risk management and progress reviews reduce frustration.

Reduces the risk of runaway projects. Studies demonstrate that management may lose patience with the development of an application and finally abandon the project out of sheer frustration. This outcome may be manifested in a project that seems to be hopelessly off track and doomed to failure. Runaway projects never end; therefore, management should consider canceling these projects. Risk can also be related to urgency. When delivery of an application is absolutely urgent, the development risk is automatically increased as urgency generally effects quicker actions, causing important activities to be missed or to be performed with less rigor. Political risk may also be involved with the project such as corporate political implications and general opposition to the new system. Rapid Delivery alleviates these elements by delivering functions significantly earlier. Risk management and progress reviews as integral components of the Rapid Delivery method reduce frustration by identifying these elements early and then providing the ability to manage them from the point where they are identified.

Diminishes the risk of never completing the project. Sometimes the risk of never completing a project stems from overstated expectations. When management and users alike believe applications will deliver more than can reasonably be accomplished, the expectation will change unless it is explicitly reset or redirected. Ambiguous project objectives always affect the risk of a project and many times lead to a project never being completed. Likewise, project objectives that start out very clearly defined and then crumble because of mismanaged changes issued against the developing application are prime candidates for never being completed. "Fuzzy" application requirements and design specifications also lead to high-risk projects; it is difficult to build something that has not been articulated clearly. Dividing the application into application segments enhances requirements gathering by identifying only the requirements and design specifications necessary to build the application by each of its sequenced segments.

Another key point is that applications that have been developed to a point can be examined on a continuing basis. Applications that should not be completed can be identified earlier, providing cost savings on projects that would otherwise be built. Flawed applications can be identified, and development resources can be redirected to projects that are perceived to be of more value to the corporation.

Lessens the risk of missing important or urgent user requirements. Avoiding or missing important user requirements is project suicide since the final recipient of the completed application is the user. Moreover, user requirements that become increasingly important must also be addressed so that they can be built into the application in a planned fashion. To complicate matters, it is often difficult to gain consensus across departmental boundaries when several departments are to provide requirements for the application. These requirements are often inconsistent and conflicting. Rapid Delivery provides the ability to manage these requirements by gathering user needs and design specifications at the time when each application segment will be built, thus providing the ability to manage changing requirements. Additionally, as the users gain experience with the application and its growing capabilities, they will begin to find new ways to use it and discover new application requirements to support these uses that otherwise might not be specified as part of the

application development by using a monolithic approach.

Encourages concepts that make the project adaptable. Sometimes projects fail for reasons related to adaptability. When enterprise reorganizations take place, projects are frequently impacted by changes in priorities and the level of importance with which projects are viewed by the individuals in the changed organization. Project objectives that frequently change also require projects to have the ability to adapt to the situation and survive. Business changes also may impact the way business is conducted as well as the pace of the changing business environment. A secondary point in project adaptability is in the area of technology. Since Rapid Delivery focuses on evolutionary principles, technological changes can be injected into the development effort as appropriate. Rapid Delivery incorporates risk management and progress reviews, which keeps the application flexible in changing business and technology environments. Rapid Delivery capitalizes on change by making well-managed change advantageous so as to provide users with the functional capabilities they want in a timely fashion. Finally, when new application segments are integrated with existing ones, potential performance problems can be more easily identified. If the application performs well before a new application segment is integrated, problem determination time can be reduced, as the most recent application segment is the most likely culprit.

Reduces the risk of underestimating effort. Since Rapid Delivery does not attempt to totally define the application requirements at the beginning of the development project, the risk of underestimating the amount of effort required to produce the application is reduced. Segmenting the application development effort permits a better estimate of the necessary effort. Additionally, as increasing skill is built into the development of large projects, the increased skill can be used in future development effort estimating. Since each application segment is estimated separately, the risk of underestimating the effort should be greatly reduced.

Enhances application completion flexibility. When the application matures to a certain point, it may be advantageous to discontinue the project when sufficient functionality has been delivered. This means that the application can be developed only to the point of benefit and not beyond, as would be the case with traditional development approaches. In addition, application segments that have not been started can be resequenced if appropriate.

Increases I/S responsiveness and credibility. As each application segment is delivered, the user community will see the I/S organization as increasingly more responsive. I/S credibility will be boosted since application segments and user functional capabilities are delivered in a regular, scheduled manner. Application functionality is increased with each delivery, and schedules, which were previously not met, will be seen as project milestones to be looked forward to.

Provides an early capability to train users. When each application segment is being tested, the capability to bring in users to assist in the testing exists. Use this early application availability to also train the users on each functional capability that evolves over time. In this way, the users can more easily absorb and internalize the application as opposed to traditional development patterns in which the users must master the entire application all at one time.

Improves the installability of applications. As each application segment is defined and developed, the ability exists to install smaller, more manageable components of the application. These smaller components make the installation and integration easier, faster, and less trouble-prone. It also makes better use of I/S resources.

Considerations for this method

The evolutionary approach may lead some individuals to attempt to define every thinkable application function into the first application segment. This phenomenon occurs in general because individuals unfamiliar with the evolutionary approach may fear that the application will be terminated early or be abandoned, or the approach discontinued. Ensure that each application segment contains reasonable numbers of functional capabilities.

Rapid Delivery is most favorably used with the development of applications that do not presently exist. This observation can be attributed to the segmented delivery of applications. When Rapid Delivery is used to replace portions of an already

existing, complete application, "trestles and bridges" must be built between the old and the new application to provide all of the capabilities available within the existing software as well as providing access to the replacement functions in the new software. These trestles and bridges may be as simple as a single interfacing program between the two applications, to a series of programs that must be written to accommodate the entire spectrum of user capabilities in both the new application and the old application. Later, as functions are added to the new application, the trestles and bridges can be removed.

When the system is designed, consider the durability of the application. If a bad design decision is made or an inadequate application architecture is defined, it will be difficult to build the remainder of the application. Modifying additional application segments to fit a bad design or architecture will trigger additional application maintenance later to force inconsistent application components to fit. This application maintenance is one of the fundamental elements that Rapid Delivery attempts to address.

Ensure that there are individuals within the organization who have been trained in risk assessment, risk analysis, and risk management. The risk management process can also be used in the training and education process. As project risks are assessed and new risk exposures are discovered, the information should be documented and communicated across the development organization. In this way, undesirable risks can be avoided.

It is not advised that Rapid Delivery projects be staffed with inexperienced, or junior-level, development personnel. To do so may cause many of the problems that Rapid Delivery attempts to alleviate.

Summary

We have described an evolutionary approach that can be applied to large-scale application development projects and review the key points here.

Rather than attempting to develop the application as a single, large project, requirements are gathered at a high level during the high-level requirements activity to break the application into application segments. During the division, an application segmentation strategy is selected. The application is segmented and is followed by ordering the segments into an overall development sequence. The completion of each of these projects provides the ability to regularly inject increasing functionality into the delivered application base, having it evolve over time.

Application segment development uses standard development methods and techniques but adds iteration. In this way it takes advantage of existing skills within the enterprise while reducing the need for expensive training and education. Application segment delivery takes place at the completion of each application segment. Once completed, each application segment is integrated into the existing environment and is implemented in the production environment during the integration/implementation activity.

Ongoing activities decrease the overall risk of the project and improve on development schedules. Risk management provides the ability to manage risks from the point of identification. Progress reviews track the status of the overall project at regular intervals rather than more traditional "phase exits"—phases can no longer be identified for the overall projects.

Cited references and notes

1. Personal communications with anonymous client sources. These data were gathered through client interviews and conversations.
2. P. Atkinson, "Ten Top Shops," *Canadian Datasystems* **22**, No. 9, 34, 39 (September 1990).
3. L. H. Young, "Product Development in Japan: Evolution vs. Revolution," *Electronic Business* **17**, No. 12, 75–77 (June 1991).
4. Anonymous, "What Makes Yoshio Invent?," *Economist* **318**, No. 7689, 61 (January 12, 1991).
5. P. R. Melichar, "Management Strategies for High-Risk Projects," *IBM Information Systems Management Institute*; available from the IBM Consulting Group.
6. D. McComb and J. Y. Smith, "System Project Failure: The Heuristics of Risk," *Journal of Information Systems Management* **8**, No. 1, 25–34 (Winter 1991).
7. J. Rothfeder, "It's Late, Costly, Incompetent—But Try Firing a Computer System," *Business Week* (November 7, 1988), pp. 164–165.
8. M. A. Griesel, *Incremental Development and Prototyping in Current Laboratory Software Development Projects: Preliminary Analysis*, JPL Publication 88-41, NASA Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA.
9. *Object*, in this sense, is intended to be analogous to the user interface object-action approach. We define an object as being a *thing* that is to be manipulated. *Actions* are performed upon an object.
10. F. Land, "Adapting to Changing User Requirements,"

Information and Management, North-Holland Publishing Co., New York, No. 5, 59–75 (1982).
11. W. B. Foss, "Software Piecework," *Computerworld* (September 23, 1991), pp. 69–70, 74.
12. U.S. Air Force, *Adapting Software Development Policies to Modern Technology*, National Research Council Commission on Engineering and Technical Systems, Washington, DC (1989).

General references

L. J. Arthur, *Rapid Evolutionary Development, Requirements, Prototyping & Software Creation*, John Wiley & Sons, Inc., New York (1992).

L. J. Arthur, *Software Evolution—The Software Maintenance Challenge*, John Wiley & Sons, Inc., New York (1988).

B. W. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer* **21**, No. 5, 61–72 (May 1988).

D. R. Graham, "Incremental Development: Review of Nonmonolithic Life-Cycle Development Models," *Information and Software Technology* **31**, No. 1, 7–20 (January/February 1989).

E. V. van Horn, "Software Must Evolve," *Software Engineering*, H. Freeman and P. M. Lewis, Editors, Academic Press, New York (1980).

D. L. Hough, *Rapid Delivery, Incremental Development, and Prototyping: Methods for Responsive Applications Development*, IBM Consulting Group, IBM Corporation (1993); available from the IBM Consulting Group.

Accepted for publication March 15, 1992.

Donald Hough *IBM Consulting Group, 1605 LBJ Freeway, Dallas, Texas 75234.* Mr. Hough is a consultant within the IBM Consulting Group, specializing in application development methodologies. He has developed methodologies for both IBM and other clients and has served as consultant in the use of methodologies, their implementation, cultural and organizational impact, and accompanying education. He has also managed application development efforts in the financial, telephony, oil and gas, government, research and development, and transportation industries. Mr. Hough has developed methods for Rapid Delivery, Incremental Development, and Prototyping for use by IBM and its customers worldwide. In addition, he has authored an IBM application development methodology book, *Rapid Delivery, Incremental Development, and Prototyping: Methods for Responsive Applications Development*. He has spoken to a variety of professional groups on methodologies, evolutionary development, and prototyping.

Reprint Order No. G321-5518.