

--- Copyright © 1998, Robert G. Fichman and Scott A. Moses ---

**AN INCREMENTAL PROCESS FOR SOFTWARE IMPLEMENTATION**

**Robert G. Fichman, Ph.D.**

Assistant Professor  
Boston College, Carroll School of Management  
452B Fulton Hall  
Chestnut Hill, MA 02167-3808  
Ph: 617-552-0471  
Fax: 617-552-0433  
fichman@bc.edu

**Scott A. Moses, Ph.D.**

i2 Technologies  
909 East Las Colinas Boulevard, 16<sup>th</sup> Floor  
Irving, Texas 75039  
Ph: 214-860-6000  
Scott\_Moses@i2.com

Published in *Sloan Management Review* Vol. 40, No. 2, 1999 pg. 39-52.

## AN INCREMENTAL PROCESS FOR SOFTWARE IMPLEMENTATION

### Abstract

A results-driven incremental approach to implementing advanced software technologies promotes organizational learning, sustains implementation momentum, and avoids the delay, cost and risk of over-engineered solutions.

### 1. INTRODUCTION

Eighteen months had elapsed since New World Electronics began implementing an advanced software-based scheduling system for the factory floor.<sup>1</sup> From the beginning, all the traditional elements were present for a smooth implementation: support of senior management; adequate staffing and funding; a good fit between the needs of the organization and the capabilities of the software; and a solid information technology (IT) infrastructure on which to build. And yet, after a vigorous start, the project team had somehow lost its way. What had seemed to be a relatively straightforward task—configuring desired software functionality and specifying complementary organizational changes—turned out to be surprisingly burdensome. As the team engaged in cycle after cycle of analysis and configuration, they seemed to continually approach but never quite reach a complete and sound model tailored to their manufacturing environment. Milestone dates came and went, and resources and attention started drifting away. The team was left wondering whether anyone or anything could refocus the energies of the organization on their task.

When it comes to implementing advanced software technologies, New World Electronics is not alone.<sup>2</sup> The days of “turn-key” packages—technologies that can be physically installed, turned on, and used as-is by most organizations—are over, if indeed they ever truly existed to begin with. Virtually all advanced production technologies today—CAD/CAM, executive support systems, digital imaging, work flow management, enterprise resource planning, GroupWare—either have a substantial software component, or come entirely embedded in software.<sup>3</sup> The broad flexibility of modern software can be both the boon and bane of technology implementation. This flexibility enables fundamentally new ways of working that amplify the potential benefits of IT investments, but at the cost of making almost any implementation project a potentially risky program of organizational innovation and change. It allows users to pick and choose from an abundance of functionality, but at the cost of imposing upon them the burden of choosing wisely—that is, designing a software configuration that is not only internally consistent, but also complements existing and planned organization processes, policies, structures and measures. Moreover, this flexibility enables new strategies for managing the process of implementation itself, but again, at the cost of choosing wisely from among alternative strategies.

The root of the problem at New World Electronics was not a poor endowment of resources or some other project *factor*, but rather, an ineffective implementation *process*.<sup>4</sup> Team members employed a traditional approach to software implementation, characterized by a lengthy period of off-line effort to select and configure software functionality and concluding with a “big-bang” finish—a concentrated interval during which the complete system would be turned on, and the organization would make a clean, one-time cutover from the old systems and practices to the new. In this instance, the big finish never arrived. All-at-once implementations are sometimes

unavoidable—as in the case of a completely indivisible technology—but this is usually not the case for technologies embedded in software.<sup>5</sup>

In this article, we present a strategy for guiding the implementation of advanced software technologies based on the principle of *results-driven incrementalism* (RDI). The inspiration for this strategy arose from observations of the problems experienced on traditional software implementations at New World Electronics and elsewhere.<sup>6</sup> The strategy described herein bears little resemblance to the gradual pace and deemphasis on specific results commonly associated with incremental approaches.<sup>7</sup> Rather, it specifies that a project be divided into a series of short, intensive cycles of implementation, each of which delivers a measurable business benefit. We have found this approach promotes organizational learning, maintains implementation focus and momentum, and negates the common tendency to over-engineer technology solutions—all of which serve to substantially shorten the time to the arrival of business benefits and to reduce the risk of implementation failure.

Innovation researchers and software methodologists alike have long advocated incremental approaches to technology implementation (at least in the right circumstances), and we believe most managers accept the wisdom of incrementalism in the abstract. Yet on actual projects it's rarely practiced. We think this happens for a variety of reasons—managers may not understand what incrementalism really means or how to use it, or may view it as providing marginal value, or may believe it is infeasible in their specific case. Aside from presenting the RDI methodology itself, this article provides specific guidance on how to counter these kinds of inhibitors.

We begin by addressing the question of why an explicit process model is needed to manage the implementation of advanced software packages, and then present the main elements of a results-driven incremental strategy. We then describe a specific application of the strategy during the implementation of supply chain planning and scheduling software at Herman Miller, a large manufacturer of office furniture systems. This example serves as a concrete illustration of the method itself and its typical challenges and benefits.<sup>8</sup> Finally we discuss two key issues raised by the specifics of the implementation at Herman Miller and elsewhere: (1) why the methodology is so often resisted, and (2) what factors are critical to achieving success with the methodology.

## **2. THE EVOLVING NATURE OF SOFTWARE IMPLEMENTATION**

Two elements still characterize many software package implementation projects despite being increasingly out of step with organizational realities. The first element is a narrow focus on the functions of the software itself, where the goal of implementation is viewed as a straightforward delivery to the user population of some particular set of software functionality. The second element is a penchant for all-at-once delivery scenarios, where the implementation team conducts its work off-line, often for a year or more, and then endeavors to put the entire software configuration into use at once.

In years past, software packages tended to incorporate more rigid assumptions about organizational structures and processes. However, in recent years all this has changed. No longer a tool to simply automate and speed up current ways of working, advanced software is now seen as a means to enable fundamentally new policies and work organizations.<sup>9</sup> A consequence of this conceptual shift has been to complicate the task of software package implementation teams. Never an easy task, software implementation has now taken on more of

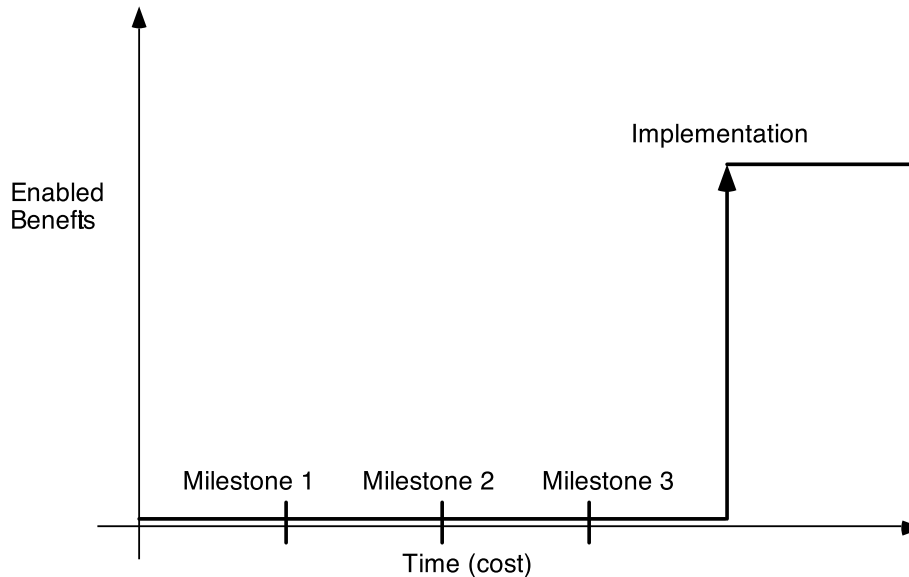
the character of technological process innovation (sometimes revolutionary, other times evolutionary)—with the attendant risks and challenges of such initiatives.

Many have noted that the implementation of process innovations inevitably involves learning and adjustment costs, the magnitude of which often exceed the raw purchase cost the technology itself.<sup>10</sup> Technological process innovations can rarely be implemented “as is,” so to achieve desired results adopters must engage in cycles of adaptation where the technology and organization are fit to one another.<sup>11</sup> The software configuration challenge then consists of a process where implementers develop an understanding of the software itself and what organization design options the software enables and then work to actually configure, deliver, and continuously evolve the software together with corresponding organizational changes.

It is easy to understand how organizations might systematically underrate the magnitude of this challenge. Many organizations turn to packaged solutions in the first place because of the not unreasonable belief that what’s difficult about software development and implementation has been “packaged up” by the software vendor. Ironically, however, as packages have become more sophisticated and flexible, the act of configuring and implementing software has begun to assume some of the complexity that used to be associated with developing a custom system. Advanced software packages typically provide hundreds or thousands of discrete features and data items that may or may not be used, and when used can behave in multiple ways. Technical writers and implementation consultants can describe these features in isolation, and might be able to summarize a few standard or typical configurations. But they can not anticipate—from the seemingly infinite number of possible combinations—the most effective configuration to achieve particular results, and they can not anticipate how any given configuration will interact with a particular organizational context. As a result, organizations implementing advanced software packages need an effective model that guides the process of discovering and implementing an effective initial software configuration.

### **3. TRADITIONAL VS. RESULTS DRIVEN INCREMENTAL SOFTWARE IMPLEMENTATION STRATEGIES**

The results-driven incremental (RDI) strategy was developed after observing pitfalls of more traditional functionality-driven, all-at-once implementations. This traditional implementation strategy is depicted graphically in Figure 1. The horizontal axis represents time (and may also be viewed as a rough proxy for implementation cost) and the vertical axis represents the level of business benefits enabled by the implementation.

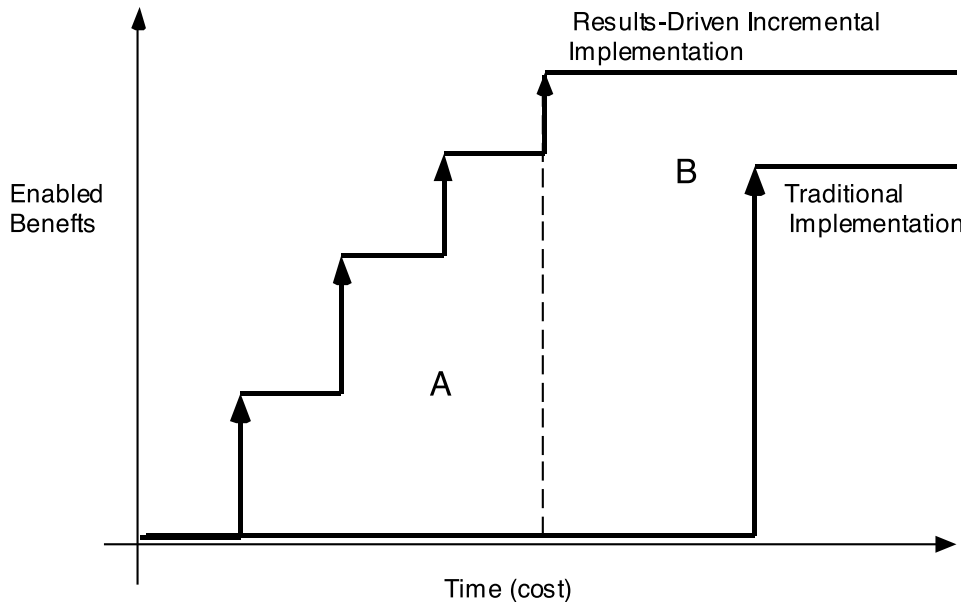


**Figure 1: Traditional Software Implementation**

The traditional approach not only delays the arrival of business benefits until the end of the project, but has other important pitfalls as well. It allows, and in fact encourages, implementers to focus on technology itself instead of the corresponding organizational changes required to actually derive value from implemented functions. In addition, the absence of clear linkages to business value invites “over-engineering,” that is; implementing functions that may never be used or adding excessive details that are not in fact necessary to achieve desired business results. Over-engineering obviously wastes resources, but perhaps more importantly, it produces an explosion of complexity as team members must figure out how each feature works, and then anticipate how it interacts with other software features and with the policies and procedures of the adopting organization. As one participant in the Herman Miller implementation put it:

When the difference between where you are going and where you are now is great, it is painful for the production floor... If it gets out on the floor and people choke they will say “we can’t get orders out to customers, turn it off.” Usually they are choking on something that people who designed and implemented the system hadn’t anticipated. [Operations VP, Herman Miller]

The RDI strategy stands in stark contrast to the traditional approach. With this strategy, targeted business results drive the entire implementation process—from the articulation of goals at the beginning of the process, to the tactical decisions made during implementation, to the evaluation of project success during and after implementation. A plan is devised that divides system delivery into a series of short (i.e., two to three month) increments, each of which delivers a combination of software functionality and organizational change. Each increment is defined so that intended business results will be achievable, *even if no further increments are completed*. (This requires that the underlying technology possess at least moderate divisibility, however this will be true of most software technologies.<sup>12</sup>) For each delivery increment, specific measures are defined to gauge improvements in business performance, and operational decisions made throughout the implementation process are guided by their impact on these targeted improvements. This approach to software implementation is contrasted with traditional approaches in Figure 2.



**Figure 2: Results-Driven Incremental vs. Traditional Implementation**

The most obvious advantage of the RDI approach—one shared by incremental approaches more generally—is simply that the stream of business benefits arrives much sooner. Graphically, this benefit is represented by area A in Figure 2. However, implementers following the RDI approach have found that it not only compresses the time to getting *some* benefit, but it dramatically shortens the time to complete the entire initial implementation and increases the overall level of project benefits. These additional benefits (represented by area B in Figure 2) arise from combining incrementalism with a strong focus on business results—a combination that has startling effects on organizational learning and implementation momentum.

We now present some specifics of a methodology that achieves a harmonious pairing of these two elements, and then describe how Herman Miller, a manufacturer of office furniture systems, used this methodology to guide a major implementation of manufacturing software.

#### **4. DRIVING INCREMENTAL IMPLEMENTATION WITH RESULTS: SPECIFICS OF THE RDI METHODOLOGY**

Before an implementation strategy can be effectively employed on actual projects, it must be elaborated into a step-by-step procedure or *methodology*, and it must be tailored to the specifics of the technology to be implemented. In this section, we describe an RDI methodology developed to guide the implementation of supply chain planning and scheduling software.

The RDI methodology incorporates five key principles:

- 1) Use targeted business results to drive decision making throughout the implementation process;
- 2) Divide the implementation into a series of non-overlapping increments, each of which enables measurable business improvements even if no further increments are implemented;
- 3) Ensure each increment implements *everything* required to produce desired results, i.e., software functionality *and* complementary changes to organizational policies, processes, structures and measures;

- 4) Size the increments so that each can be implemented in a short time (ideally three months or less);
- 5) Use the results of each increment as a basis to flesh out and adjust the plan for subsequent increments.

In the RDI methodology, the incremental segments of an implementation are referred to as *Business Releases*. This term is derived from the idea that, just as software vendors will typically deliver packages to the market as a series of software releases, an implementation of software at a particular customer site may be viewed as a series of “releases” of software enabled business results. Each Business Release consists of an integrated set of software functionality and organizational change that enables a specific, measurable business improvement in some *key performance indicator* (KPI). Typical KPIs for sites implementing supply chain planning software include due-date performance, planning cycle times, order lead times, throughput, and inventory (raw, work-in-progress and finished).

Business Releases are short and do not overlap. A typical time frame appropriate for supply chain planning software is two or three months, although this may vary depending on the nature of the software being implemented. Some organizations may be tempted to pack most of the implementation into one Release, or to compensate for delays on a current Release by concurrently initiating the next one. However, long or overlapping segments defeat the purpose of the methodology. Long segments invite the same problems that accompany all-at-once implementations; over-lapping segments work against the goal of providing isolated, cumulative episodes of learning.

The methodology begins with a Business Analysis step that defines the contents of the first few Business Releases. In the Business Analysis step, the project team seeks to answer the following questions: Which are the important KPIs to target? What are the key constraints limiting each KPI? What are the software functionalities and corresponding organizational changes that would improve each KPI? How much improvement is possible? The result is a description of each Business Release covering the following elements.

- The targeted business result (e.g., improved due date performance);
- A list of software functionalities to be implemented (e.g., automated capacity optimization);
- A list of complementary changes to organizational policies, procedures, measures, and structure (e.g., revision of the relative priorities on due date performance versus machine utilization);
- A metric for measurement of business results (e.g., on time order percent).

A template is used to capture the high level contents each Business Release (see Figure 3). The various implementations at Herman Miller each required four to six Business Releases; this has been typical of RDI implementations more generally. The Business Analysis step identifies the first few Business Releases at a high level, and defines the first Release in detail. The details of subsequent Releases are defined in “just in time” fashion following completion of the prior release.

Site Name: \_\_\_\_\_

Key Performance Indicator:	_____
Amount of Improvement:	_____
Summary of New Capability:	_____
Software Functionality to be Implemented:	_____
Changes to Data:	_____
Changes to Processes:	_____
Changes to Performance Measures:	_____
Changes to Organizational Structure:	_____
Implementation Start Date:	_____
Estimated Completion Date:	_____
Explicit Measure of Business Results:	_____
Frequency of Measurement:	_____

**Figure 3: Business Release Template**

An important consideration in devising the contents of each Business Release is the expected sequence of implementation. The ideal sequence promotes multiple objectives: it starts out in an area where conditions are most favorable to success; provides for a pattern of experimentation and learning that is most cumulative; and addresses opportunities with the highest potential payoff earlier rather than later. Naturally, these objectives will often be in conflict, thus requiring tradeoffs to be made based on management priorities at the site.

Following the Business Analysis step, the implementation team begins the process of completing the specified Business Releases. A Business Release is considered complete only when software functionalities are in actual production use, organizational changes have been made, and measurement of results has begun. All three elements are essential. Unless the software is used in production, it has not truly been implemented; if complementary organizational changes have not been made, the software will (at best) be automating outmoded practices; if measurement of results has not begun, there will be no objective basis for the evaluation of results.

Following the completion of each Business Release the project team performs a number of activities: the results of the Business Analysis step are reviewed for validity, the content of the next Business Release is defined in detail, and a new Business Release is added to the queue such that a rolling set of Business Releases is maintained. In this way, the methodology achieves the goal of maintaining an overall plan while providing for the incorporation of new information as the implementation unfolds.

### **5. AN ILLUSTRATION: IMPLEMENTATION OF SCHEDULING SOFTWARE AT HERMAN MILLER**

To further illustrate the RDI approach and its potential benefits, we describe the application of this methodology as it was employed at Herman Miller. Herman Miller Inc. is the second largest office furniture manufacturer in the United States, processing more than 3,000 customer orders per week. The company produces a full line of products (chairs, desks, walls, partitions, etc.) with a focus on providing complete office systems. Approximately 80% of the company's business is project based, and fulfilling orders requires complex coordination among multiple

raw materials and assembly plants. It is not unusual for a single order to involve five different plants. The product lines are highly optioned, with large swings in product mix.

In the mid 1990's, when Herman Miller first began looking for planning and scheduling software, the company had been facing an increasingly competitive business environment. Expected order cycle times had dropped from 12 to 4 weeks, and discounting was on the rise. Policies and operating procedures that had served them adequately in the past were now leading to deteriorating due date performance and excessive costs of goods sold. Batch processing of orders led to loss of customer identity in production—managers could not see the impact of changes or disruptions on one order, or on all orders together. As a result, it was not unusual for late availability of a small component to cause hundreds of thousands of dollars worth of production to sit in inventory. Consolidation of orders in distribution centers prior to shipment added to cycle time delays and inventory costs.

In the end, the Herman Miller search team judged the planning and scheduling software package provided by i2 Technologies to be well suited to addressing the above problems. Over the past two years, Herman Miller has implemented the software in six sites.

When pitching the idea of using a results-driven incremental approach, implementation consultants found a receptive audience from key managers who had personally witnessed the pitfalls of functionality-driven, all-at-once implementations at Herman Miller and elsewhere. Nevertheless, several Herman Miller employees were resistant to using the methodology:

The methodology is different than what most people are used to. It can be a very frustrating process, the initial planning sessions...people will not always want to think through their operational goals and opportunities. They want to jump right in and start configuring capabilities in the software....Staff feel uncomfortable thinking in terms of business goals rather than software capabilities...they feel safer thinking in terms of activities rather than in terms of achieving business results. [Implementation project manager, Herman Miller.]

In the end, managers favoring the methodology were able to persuade resisters to reserve judgment until completion of the first incremental portion of the implementation. Positive results on this first increment brought the resistant team members around: "It's like it created 'draft'—you know, how this happens in racing—it created draft and pulled people in" [Operations VP, Herman Miller].

## 6. IMPLEMENTATION RESULTS

The implementation at Herman Miller delivered new software capabilities, including retention of customer identity throughout the manufacturing process, use of constraint-based scheduling at the plant level, and global synchronization of all the plants involved with an order. Table 1 summarizes the contents and sequence of implementation at Herman Miller's Chair plant as they actually occurred.

**Table 1: Overview of Herman Miller's Business Releases**

BR	KPI	Description	Schedule	Software Functionality to be Implemented	Organizational Changes to be Implemented
----	-----	-------------	----------	--	--

1	Planning Cycle Time	Identify current material shortages	12/4 - 12/22	Detection of errors in data model Rapidly create planned material assignments Identify demand that is affected by current material shortages	Incorporate new planning tool in planning process Make corrections to missing or inaccurate data
2	Due Date Performance	Create material feasible schedules	1/3 - 2/23	Suggest material procurements considering supplier constraints Synchronize material procurements for an order tree Identify demand that is affected by future material shortages Simulate effects of possible expediting actions to resolve material shortages	Decide which materials should be controlled Only purchase a material if all materials for an order can be purchased Implement consistent buying practices for each product group Create a reaction plan for use when a quality problem occurs with incoming material
3	Throughput and Manufacturing Lead Time	Create capacity and material feasible schedules	2/26 - 4/19	Model true plant capacity with routing times for each operation for each material Release three sequences of work per day (instead of one per week) Identify and resolve capacity overloads Create production sequence for the whole plant Synchronize capacity plans with material plans	Implement consistent scheduling practices for each production line Schedule based on objectives of the whole plant rather than of individual departments Only release orders that can be fully completed Use the planning tool to determine capability to accept short lead time orders
4	Direct Shipment of Orders from Plant to Customer	Synchronize manufacturing with customer orders	4/22 - 6/14	Modify transaction system to supply customer order data to plant Synchronize completion times of all line item orders for a customer order Maintain pegging of line item orders to direct ship customer orders after quantities have been consolidated Use a "smart ticket" for production tracking	Change emphasis from machine utilization to due date performance Define process to capture customer "need date" Create a "red flag" plan for use when a line item order is behind schedule

However, one of the most striking elements at Herman Miller was the willingness to embrace complementary process and policy changes:

We've seen improvements in almost every key operational metric at Herman Miller. But I want to clarify that the results have been driven by major process and policy changes as

well as by the [software] implementation. Don't ever think just plugging in the software by itself will do it for you. [Implementation project manager, Herman Miller.]

These changes included radical modifications to their order change and cancellation policies, discontinuation of certain product lines that were most responsible for excess raw materials inventory, changes to how lots were sized and how orders were consolidated prior to shipping, and perhaps most importantly, changes to their whole philosophy of measurement. Prior to the implementation, Herman Miller had a measurement adverse culture, and those few performance measures that did exist conformed to such traditional objectives as maximizing machine utilization. Over the course of the implementation, the software enabled a new and pervasive set of measures to be defined and implemented.

It's been really interesting watching the change take place over the last couple of years...before, people wanted to celebrate good *effort*, and were not as focused on [the idea] that celebration belongs to good *results*. [Implementation project manager, Herman Miller.]

Having completed several implementations using the RDI methodology, Herman Miller has become a strong proponent of the approach. With the exception of one site where resistance to the methodology was particularly high, each site completed the first increment within six months and the entire implementation within a year. (In the site where the methodology was resisted, implementation took 18 months.) The full implementation across all six sites was completed on time and within budget. In terms of key performance indicators, Herman Miller reported achieving a 75% reduction in planning cycle time, a 22% decrease in lead times, a 79% increase in inventory turns, and a decrease in missed due-dates from 30% to 1%.

**Table 2: Implementation Site Comparisons**

	Sites Employing the RDI methodology (N=10)	Sites Not Employing the Methodology (N=18)
Median time to first production use	5 months	8 months
Median time to project completion	10 months	> 20 months
Median % of implementation goals achieved within 18 months	90%	40%

Employees at Herman Miller attributed their positive i2 implementation experience to a large extend on the *organizational learning* and *implementation momentum* that the RDI methodology fostered. This emphasis on learning and momentum is consistent with prior research on technology implementation, organizational learning, and the management of task-focused teams.

### **Organizational Learning Impacts of RDI**

As with complex technological innovations more generally, the burden of organizational learning represents a fundamental challenge to the successful implementation of advanced software packages. With traditional approaches, crucial mechanisms for effective learning—those based on everyday use of the technology—are either absent, or are deferred until a year or more into the implementation effort. Organizational learning theorists have argued that effective learning tends to be *incremental*, *intensive*, *immediate*, and *action-oriented*. It is *incremental* because new knowledge is most easily absorbed when it can be layered on top of existing knowledge.<sup>13</sup> It is *intensive*, because learning usually does not “take” until concepts have been mastered, and drips and drabs of interrupted effort rarely lead to such mastery.<sup>14</sup> It is *immediate*, because learning is

elusive when effects are separated from causes by space or time.<sup>15</sup> It is *action-oriented*, because only through action do people build up the vast storehouses of tacit knowledge that form the foundation of any skilled performance—whether playing the violin, or operating a transformed organization in the wake of a major software implementation.<sup>16</sup> All of these mechanisms are present in a results-driven incremental implementation.

Business Releases create a very solid, concrete framework in which people have to achieve certain results. Those results then drive what they go out and try to learn. [Operations VP, Herman Miller.]

You can show people examples, but I think that until you affected their work directly... you got that look like ‘well that’s really interesting’ ...But when these people actually started to see that they were shipping orders and that product was flowing through the plant better...We now had disciples. They were convinced. [Operations VP, Herman Miller.]

A large portion of the effort of implementation consists not of physically configuring software functionality, but of learning what business process options are enabled by the software tool. Typically, users do not truly understand advanced software until they have attempted to use it to solve real problems with live data:

It is easy to tell someone “you don’t need that inventory buffer”. Many consultants have come through telling them they don’t need the buffers...but until they could see it for themselves, they are not willing to make the change. [Implementation project manager, Herman Miller.]

The individual increments of the RDI methodology provide recurring episodes of organizational learning. Each increment also lays a foundation of knowledge for subsequent increments, and often results in the identification of alternative approaches that may alter the direction of remaining increments for the better. Traditional implementations, by contrast, attempt to aggregate all hands-on learning into a single, massive delivery cycle. As a result, all of the inevitable bugs and mismatches between the technology and the organization that were accumulated during the off-line portion of the process get encountered at once.<sup>17</sup> Traditional implementations can therefore be likened to a medical study where many different treatments are applied to subjects at the same time.<sup>18</sup> The RDI approach, by contrast, can be likened to a series of single treatments, where the result of each treatment is evaluated separately:

It allows key concepts to be tested in isolation. Where you’re talking about specific actions that you believe will achieve a specific results...they won’t get lost in all of the other things that are happening at the same time [in a traditional implementation]. [Implementation project manager, Herman Miller.]

In summary, the RDI methodology (and results-driven incremental strategies more generally) provide the following learning related benefits:

- The use of multiple increments provides an expected sequence of learning, and divides learning into discrete, manageable segments;
- The explicit performance targets help the implementation team determine what learning is relevant (i.e., which features of the technology and organization must be investigated) and thereby reduces the tendency to over-engineer the solution;
- Direct observation of results occurs regularly, and in proximity to the actions that produced the results;

- The implementation team must intensively learn what they need to know about each increment due to the short time frame for each increment;
- The completion of each increment lays a foundation of knowledge for subsequent increments.

### **Implementation Focus and Momentum Impacts of RDI**

As any project manager (or participant) knows, the intensity of effort on a project increases in the vicinity of a deadline. Informally, this is called the “deadline effect.” On the other side of the coin, it has often been observed that when slack exists in a schedule, work will usually expand to fill the available time rather than the project or task being completed early. This is known as “Parkinson’s law.” A recent simulation study by IBM has shown that when reasonable assumptions are made about the magnitudes of the deadline effect and Parkinson’s law, teams of software developers will exhibit similar levels of schedule conformance on a given task when given the same collection of milestones dates—*even when the teams differ significantly in terms of staff productivity*.<sup>19</sup> Similar ideas run through Connie Gersick’s classic study of the affect of time awareness on group behavior.<sup>20</sup> Specifically, she found that task-focused teams exhibited bursts of energy and progress at the approach of two temporal milestones—at the completion date of a project, and more interestingly, at a point precisely half way through a project. In between, project teams exhibited periods of inertia characterized by sporadic, unfocused effort. Her major conclusion was that a team’s progress was triggered “more by an awareness of time and deadlines than by completion of an absolute amount of work in a specific developmental stage.” These studies suggest that timing and nature of temporal milestones may be as important in terms of maintaining implementation progress as the absolute magnitude of staff resources devoted to the effort.

The RDI methodology calls for a schedule of concrete milestones at two or three month intervals, namely, the deadlines for delivery of each increment. Care must be taken to ensure that the amount of work to be done on each increment is feasible to accomplish within the given time frame. Just as allowing too much slack invites wasting of resources, schedules that are too aggressive invite cutting corners and other counter-productive measures.

One of the unexpected results of using this approach were the positive effects of these deadlines had on implementation focus and momentum. The periods of inertia that characterize typical projects simply were not present on software implementations that used Business Releases:

Certainly something that I have seen in the past at Herman Miller is people will mentally “check-out”—they continue to show up everyday—you know, show up to the meeting room—but they are mentally checked-out, and emotionally checked-out, and not working to drive that change through any more. That’s what I’ve seen in the past, and that’s what we didn’t see this time, because people were getting a sense that this project was moving [Implementation project manager, Herman Miller.]

Because people feel the results so much more quickly and so much more often they don’t fall into that kind of lull part way through the implementation where it feels like nothing is happening...it really does help to keep the focus. [Implementation project manager, Herman Miller.]

Another benefit of the short duration of each increment is it causes voluntary scope control by all team members. Usually the primary domain of the project manager, the RDI methodology

encouraged the entire team at Herman Miller to take an active role in scope control. Furthermore, it helped to empower the project manager to resist attempts to expand the scope from outside the implementation team:

This has been really key for me as a project manager...I have known a lot of people that the more they find out about [the software] the more excited they get about the different capabilities that are out there...and it's all very tempting...but I have the Business Release to look at and to say "does this have anything to do with decreasing raw material inventory? That's what we are focusing on right now. What you're asking for, yes it's a good thing, yes it's an important thing, and I'm sure it's going to be tied to one of the business results that we go after in a later step..." [Implementation project manager, Herman Miller.]

The traditional approach to software implementation of course may also include intermediate milestone dates for the completion of project tasks, and these may be defined to have short durations. However, these sorts of milestones tend to be abstract, task-oriented deadlines designed to suit the control objectives of management. While better than the alternative of having no checkpoints, such milestones do not serve quite the same psychological function as completion of an implementation increment. Traditional milestones do not provide a concrete target to shoot for; they do not provide immediate feedback on results; and they do not encourage the team to seek the most efficacious means to achieving given ends. Part of the problem is that the milestone dates on a traditional project are open to too much interpretation:

On a [traditional] implementation, the goal is defined by some sort of functionality, and when you are talking about functionality it is so easy for people to twist the scope..."[manager at Herman Miller].

Another part of the problem is that the team can't help but realize that the milestone date that really counts is the one at the end of the implementation, when the software is scheduled for delivery to users. If this date is met, no one will likely care if intermediate milestone dates were missed. With a results-driven incremental implementation, by contrast, all of the milestones call for the delivery of a tangible result, namely, putting some portion of the software system into everyday use and tracking progress towards the realization of targeted business benefits. As a result, the first delivery milestone counts as much as the last. And, unlike the abstract milestones in a traditional project, the intermediate completions of a results-driven incremental implementation represent a tangible accomplishment, and provide the basis for observation of the new technologies in action. This often serves to reinforce energy and build momentum for the work ahead.

To summarize, results-driven incremental strategies provide the following benefits related to maintaining implementation momentum:

- The use of multiple, concrete deadlines with short time frames avoids the periods of inertia that afflict traditional projects;
- The use of business results as the driver promotes voluntary scope control by members of the implementation team, and empowers the team to resist scope creep originating outside the team;
- The early, recurrent achievement of visible results promote confidence and feelings of accomplishment that energize the implementation team.

### **Other Favorable Implementation Factors**

While Herman Miller implementers saw the RDI methodology as a primary contributor to overall project success, several other favorable elements were also present. The company was facing a genuine crisis in terms of competitiveness in the market place, thus leading to a general willingness to embrace change. The team had the full support of top management throughout the project. They employed a small team of implementers dedicated full time to the project, rather than having a larger staff of part-time implementers. And perhaps most importantly, the team felt it was important to “own their own outcome,” by taking the initiative in driving the implementation through, rather than relying on external consultants to set the direction and tone.

Even so, evidence from other sites spanning a wide variety of organizations and settings suggests the RDI methodology does not require the presence of all of these favorable elements to be effective. An analysis of 28 i2 implementations initiated between May 1995 and December 1996 found that sites employing RDI reported a 40% shorter median time to initial benefits, a 50% shorter median time to project completion, and twice as many business goals achieved within 18 months of project initiation (see Table 2.)<sup>21</sup> No site that used RDI experienced implementation failure—an event that occurred occasionally under the traditional approach.

### **7. WHY IS THE APPROACH SO OFTEN RESISTED?**

As just described, the results-driven incremental approach to software implementation has many potential benefits. Yet, even so, consultants employing the RDI approach have found that although the approach appeals to most executives, implementers often resist it. Even at Herman Miller—the first customer to use the methodology and now one of the strongest advocates of the approach—there was considerable resistance among some members of the project team.

We have observed five reasons for this resistance. A first reason stems from the results-orientation of the approach. Often times team members are not comfortable with the idea of having to define—and then being held to—specific, measurable goals. Instead, they would rather be judged on the more straightforward process of delivering functionality. Being measured on functionality limits team accountability to activities under the team’s direct control.

The second cause of resistance is that incremental implementation has historically been associated with a gradual, deliberate pace of change. Therefore, to many, the notion of rapid incrementalism as called for in the RDI approach evokes a skeptical response. The answer to this paradox lies in the positive effect of more frequent feedback on momentum and learning.

A third common source of resistance is a mistaken belief that full implementation can be achieved quickly and easily. If the entire implementation can be completed in six months, this argument goes, why bother with the task of defining multiple implementation segments?

The fourth reason is that an incremental approach often requires the development of work-arounds destined to be discarded in subsequent increments. These work-arounds take the form of “throw-away” interfaces to existing information systems as well as interim organizational policies and procedures. While the cost of these work-arounds will usually pale in comparison to the cost of managing the additional complexity of the all-at-once approach, work-arounds are more directly observable, and therefore may seem more wasteful.

A fifth reason for resistance is fear that the use of multiple increments may increase the chance of a mid-project termination—either because the intermediate results have been disappointing, or because intermediate completions provide a discrete opportunity for project resources to be

deployed elsewhere. This is not an unreasonable fear. Incremental implementations do allow an organization to discover an implementation is in trouble much sooner—in fact, this may be viewed as an additional benefit of the approach from the perspective of senior managers. (In contrast, by the time it has become clear that an all-at-once implementation is in trouble, a year or more may have elapsed.) Nevertheless, it's natural for the project team to have a different view of the desirability of early problem detection, and to prefer a scenario where they have more time to get a project back on track before provoking the scrutiny of senior management.

**Overcoming resistance.** Much of the accumulated wisdom for overcoming resistance to change also applies to RDI. This includes enlisting top management support, educating stakeholders about the methodology and its benefits, providing concrete examples of successful applications of the methodology and setting up a trial application of the methodology in a favorable setting. At Herman Miller, for example, buy-in of the approach was only accomplished after a senior manager asked key stakeholders to try the first increment on faith and then reevaluate their positions.

Also, the specific reasons for why the method is often resisted suggest additional tactics on how to promote acceptance of the method. A cost-benefit analysis can be used to put the costs of developing temporary work-arounds in perspective, for example, by comparing them to the benefit of quicker implementation or reduced risk of failure. Resistance arising from fear of premature project cancellation can be countered by working with stakeholders up front to define appropriate governance mechanisms for the project. This should include contracting up front for mutually agreeable decision rules in the event of problems and setbacks. In addition, a steering committee can be formed to make unforeseen decisions about the project along the way.

## 8. CRITICAL SUCCESS FACTORS FOR USING THE RDI APPROACH

Aside from the obvious requirement that managers must overcome resistance and ensure faithful adoption of the methodology, three additional factors are critical to ensuring success with the RDI approach.

First and foremost, results-driven incremental implementation requires *divisibility* of the technology itself. Divisibility is crucial because it allows a special sort of incrementalism, namely, incrementalism where each segment enables a measurable business result in itself, *even if no further segments are implemented*. Divisibility depends on the nature and design of the components comprising the software package, and also on the level of ingenuity that managers bring to bear on how the components are selected and sequenced for implementation. Therefore, before committing to an RDI approach, managers must take care to evaluate the divisibility of a software technology in the general case (e.g., what is the granularity of components, and to what extent are the components of the product designed to allow for "stand alone" operation?) and then assess the attractiveness of alternative approaches to implementation sequencing allowed by the technology.

We see divisibility as taking two alternative forms, one in which successive increments involve the same software modules but at different levels of detail, and a second in which successive increments involve different modules. Supply chain planning and scheduling products (and modeling or decision support style information systems more generally) naturally permit divisibility of the first sort, which might be likened to applying layers of paint to a canvas. The second sort of divisibility, which might be likened to slicing up pieces of a pie, may be a more natural approach for larger transaction processing style systems.

For other kinds of software a hybrid of these approaches might be feasible. In the case of GroupWare software such as Lotus Notes, for example, an organization might plan to first use it only for workgroup calendaring and email, then create databases with fairly static and standard information, then create more specialized databases with dynamic information, then allow users to create their own databases, then implement automated work flow processes. In fact, Lotus Consulting has developed a methodology called the Accelerated Value Method (AVM), that shares the basic principles of results driven incrementalism.<sup>22</sup>

We feel that most technological innovations embedded in software—or that have a large software component—can be made at least moderately divisible by the project team, even in cases where divisibility has not been designed into the product from the start. Temporary work-arounds may be required, but the cost for these should be considered in relation to the total implementation cost. However, not all technologies possess the inherent potential for divisibility, because it is not always possible to decouple modules for serial implementation. This is sometimes the case for real-time or embedded software systems (such as air traffic control) where the smallest unit of coherent implementation exceeds the scope of an individual increment.

The second critical success factor is that the key implementation challenges posed by the technology in question should be the ones RDI has been designed to address, namely, organizational learning and implementation momentum. While these barriers are quite pervasive in the processes of technological innovation, they are not always the primary barriers to be overcome. For some kinds of technologies, barriers related to achieving a “critical mass” of users,<sup>23</sup> coordination of a large number of inter-related implementation elements,<sup>24</sup> dealing with indeterminacy about what the organization can or should accomplish with the technology,<sup>25</sup> or properly aligning incentives in the wake of the implementation may be the predominate concerns.<sup>26</sup> In such circumstances, it is possible that a results-driven incremental approach will be of lesser benefit, although further research is required to determine the extent to which this is so.

A third critical success factor is that there must be a reasonably good fit between the software technology and the needs and characteristics of the adopting organization. Dorothy Leonard-Barton has noted that the “misalignments” that inevitably exist between a technology and a target organization can be large or small, and require correspondingly large or small cycles of adaptation to resolve.<sup>27</sup> We believe that the RDI strategy can be viewed as a way *to rationalize and manage a series of small cycles of adaptation*. However, if there are large misalignments—for example, that require a change to the core features of the technology or a major organizational restructuring—the approach will be less appropriate. In this case, a large cycle of adaptation will be needed that will involve a time horizon and degree of unpredictability that obviate use of short, well-defined increments. Nevertheless, once the misalignment has been addressed, it may then be possible to use the approach on the remainder of the implementation.

One question we have been asked is whether this methodology can be applied to enterprise resource planning (ERP) offerings from vendors like SAP, PeopleSoft or BaaN. We believe that except for the instance of a small company implementing a fairly well understood module, the answer will usually be no. The problem is that the componentization of these packages occurs mostly at the level of broad functional modules—human resources, financials, inventory management, production planning—and these modules, intended as they are to support entire business functions, are usually too large to be implemented in a single increment spanning two to three months. Because each module has not been designed as combination of separable

components suitable for serial implementation, the cost and complexity of establishing required intra-module interfaces and work-arounds would often times exceed the benefit to be gained from incrementalism. Or in other words, the modules have not been designed to promote divisibility. Interestingly, the trend in these packages is towards finer granularity in the componentization of modules, so in the years to come this barrier to using an RDI approach may well fall away.

## 9. CONCLUSIONS

The trend towards ever increasing flexibility and sophistication in packaged software has magnified the challenges facing software implementation teams. A favorable set of project *factors* (top management support, appropriate staffing and training, etc.), while important, are insufficient to ensure project success. Organizations also need an effective implementation *process* for managing the journey of innovation and change that increasingly accompanies the deployment of advanced software. We have described one such approach, a results-driven, incremental strategy for software package implementation. By dividing implementation into a series of self-contained segments, each designed to achieve a specific business result, this approach counters the scope creep and overengineering that so often accompany traditional implementations. In addition, the approach acts as a focusing device that promotes organizational learning and maintains implementation momentum at a high level. The use of multiple increments with short horizons provides an expected sequence of learning, divides learning into manageable segments, and permits observation of outcomes in proximity actions that produced them. The early, recurrent achievement of visible results continually reenergizes members of the implementation team.

The typical view of technology holds that it is defined by its technical features—the *technology-as-designed*. We take an alternative view, where a technology is bounded by its technical features together with the configurations in which it is typically delivered and utilized—the *technology-in-use*. While the technology-as-designed defines the range of possible uses of the technology, it is technology-in-use that determines the immediate value to the individual customers, and perhaps more importantly, determines the extent to which the technology succeeds in accumulating a critical mass of committed users.<sup>28</sup> Designing an effective software delivery process is not easy. It requires an ongoing effort—to develop systematic implementation processes, to measure customer outcomes, to document process variants for different customer situations, and to build more implementability into a product from the start (e.g., by increasing divisibility). And it requires an education and marketing effort to encourage implementers to adopt the methodology.

Determining the most effective delivery processes for a particular technology requires ongoing R&D by *someone*. It can be done by third-party consultants, although there is no guarantee they will choose to take up the task of defining a methodology tailored to a given product, or that they will bring the degree of individual attention to the process that the product might require. It can be done by end-users, but most end-users are constrained by a sample size of only one. Or it can be done by the primary vendors themselves. We suggest that most technology vendors would be well served to view effective implementation processes as a crucial element of success, and to manage investigation in this area as they would any other key area of R&D.

## ENDNOTES

1. Although “New World Electronics” is a pseudonym, this description is based on an actual example.
2. From 50 to 75% of organizations experience failure implementing advanced production technologies. See:  
A. Majchzak, *The Human Side of Factory Automation* (San Francisco: Jossey-Bass, 1988).  
Up to 70% of business reengineering implementations fail to achieve intended goals. See:  
B.J. Bashein, M.L. Markus and P. Riley, “Preconditions for BPR Success,” *Information Systems Management*, vol. 11, no. Spring, 1994, pp. 7-13.
3. J.B. Quinn, J.J. Baruch and K.A. Zien, “Software-Based Innovation,” *Sloan Management Review*, vol. 37, no. 4, 1996, pp. 11-24.
4. Until recently, the bulk of innovation research focused on implementation *factors* rather than *process*. Factor models of implementation identify individual characteristics of the technology (e.g., complexity, trialability), the organization (e.g., centralization, formalization), or the technology-organization combination (e.g., compatibility, top management support) that either facilitate or inhibit implementation success. See:  
A.D. Meyer and J.B. Goes, “Organizational Assimilation of Innovations: A Multilevel Contextual Analysis,” *Academy of Management Journal*, vol. 31, no. 4, 1988, pp. 897-923.  
C.A. Beatty, “Implementing Advanced Manufacturing Technologies: Rules of the Road,” *Sloan Management Review*, vol. 35, no. Summer, 1992, pp. 49-60.  
Process models, by contrast, describe holistic implementation strategies and processes. See:  
B.W. Chew, D. Leonard-Barton and R.E. Bohn, “Beating Murphy's Law,” *Sloan Management Review*, vol. 32, no. 3, 1991, pp. 5-16.  
E. Brynjolfsson, A.A. Renshaw and M. Van Alstyne, “The Matrix of Change,” *Sloan Management Review*, vol. 38, no. 2, 1997, pp. 22-40.  
W.J. Orlikowski and J.D. Hofman, “An Improvisational Model for Change Management: The Case of Groupware Technologies,” *Sloan Management Review*, vol. 38, no. 2, 1997, pp. 11-21.
5. D. Leonard-Barton, “Implementation Characteristics of Organizational Innovations,” *Communication Research*, vol. 15, no. 5, 1988, pp. 603-631.
6. The initial concepts for the RDI strategy were developed in 1994 by Dr. Moses and other implementation consultants working for i2 Technologies, a vendor of supply chain management software headquartered in Irving, Texas. Over the course of applying the strategy on several large implementations it was formalized into a methodology and adopted as the recommended approach for implementing the firm's software products.

7. M.J. Gallivan, J.D. Hofman and W.J. Orlikowski, *Implementing Radical Change: Gradual Versus Rapid Pace*. Proceedings of the Fifteen International Conference on Information Systems, (Vancouver, British Columbia, Canada, 1994).  
D.B. Stoddard and S.L. Jarvenpaa, "Business Process Redesign: Tactics for Managing Radical Change," *Journal of Management Information Systems*, vol. 12, no. 1, 1995, pp. 81-107.
8. In selecting a case example for use in this article, we set out the following ideal criteria: (1) faithful application of the RDI methodology, (2) a willingness to provide researcher access to key implementation personnel and documents, and (3) experience with multiple applications of the methodology. Of the 10 sites employing RDI, Herman Miller best met these criteria, and had the additional advantage of being a site Dr. Moses already knew well, having served as a lead implementation consultant.
9. S. Zuboff, *In the Age of the Smart Machine* (New York: Basic Books, 1988).  
M. Hammer, "Reengineering Work: Don't Automate, Obliterate," *Harvard Business Review*, vol. 66, no. 4, July-August 1990, 1990, pp. 104-112.  
N. Venkatraman, "IT-Enabled Business Transformation: From Automation to Business Scope Redefinition," *Sloan Management Review*, Winter, 1994, pp. 73-87.
10. Although much of the prior research on the costs and challenges of process innovation have focused on the implementation of production equipment, the same principles apply to the implementation of process innovations entirely embodied in software. See:  
R.G. Fichman and C.F. Kemerer, "The Assimilation of Software Process Innovations: An Organizational Learning Perspective," *Management Science*, October 1997.
11. D. Leonard-Barton, "Implementation as Mutual Adaptation of Technology and Organization," *Research Policy*, vol. 17, 1988, pp. 251-267.
12. Leonard-Barton defines divisibility as permitting the division of "a technology into stages or segments, each of which delivers some benefits even if no further segments are adopted." Based on a set of fourteen case studies of technological innovation involving diverse technologies and organizational settings, Leonard-Barton concludes that most technologies have the potential for divisibility—although it is common for organizations to overlook this property. See:  
D. Leonard-Barton, "Implementation Characteristics of Organizational Innovations," *Communication Research*, vol. 15, no. 5, 1988, pp. 603-631.
13. W.M. Cohen and D.A. Levinthal, "Absorptive Capacity: A New Perspective on Learning and Innovation," *Administrative Science Quarterly*, vol. 35, March, 1990, pp. 128-152.
14. W.M. Cohen and D.A. Levinthal, "Absorptive Capacity: A New Perspective on Learning and Innovation," *Administrative Science Quarterly*, vol. 35, no. March, 1990, pp. 128-152.
15. P. Senge, *The Fifth Discipline* (New York: Doubleday, 1993).
16. R. Nelson and S. Winter, *An Evolutionary Theory of Economic Change* (Cambridge, MA: Harvard University Press, 1982).

17. Even so, mechanisms do exist to facilitate learning even under the constraint of an all-at-once approach. Chew et al. identify four levels of learning—vicarious learning, simulation, prototyping, and online learning—the first three of which can be employed throughout all-at-once implementations. They note that the most effective mechanism for organizational learning is online learning—also referred to as “learning by doing”—where learning occurs as the technology is used in the ongoing operations of a business. They argue that in most cases online learning is cost-prohibitive; however, it appears this is based on an assumption that the technology is indivisible. Our position is that when a technology is divisible, and this divisibility is used to enable a results-driven incremental approach, on-line learning becomes a key mode of learning. See:  
B.W. Chew, D. Leonard-Barton and R.E. Bohn, “Beating Murphy's Law,” *Sloan Management Review*, vol. 32, no. 3, 1991, pp. 5-16.
18. Schaffer and Thomson use this same analogy in describing the problems with diffuse, non-results driven programs of change. See:  
R.H. Schaffer and H.A. Thomson, “Successful Change Programs Begin with Results,” *Harvard Business Review*, January-February, 1992, pp. 80-89.
19. T.E. Potok and M.A. Vouk, “The Effects of the Business Model on Object-Oriented Software Development Productivity,” *IBM Systems Journal*, vol. 36, no. 1, 1997.
20. C.J. Gersick, “Time and Transition in Work Teams: Toward a New Model of Group Development,” *Academy of Management Journal*, vol. 31, no. 1, 1988, pp. 9-41.
21. The data supporting this analysis was collected by i2 from a survey of customers and project managers.
22. Principles shared by RDI and AVM include: breaking the implementation into small segments, having the contents of each segment be defined by business value, and defining each increment in such away that it provides value even if no further segments are implemented.
23. E.M. Rogers, "The 'Critical Mass' in the Diffusion of Interactive Technologies in Organizations," in *The Information Systems Research Challenge: Survey Research Methods, Volume 3*, ed. K. L. Kraemer, J. I. Cash and J. F. Nunamaker (Boston: Harvard Business School Research Colloquium, 1991).
24. E. Brynjolfsson, A.A. Renshaw and M. Van Alstyne, “The Matrix of Change,” *Sloan Management Review*, vol. 38, no. 2, 1997, pp. 22-40.
25. W.J. Orlikowski and J.D. Hofman, “An Improvisational Model for Change Management: The Case of Groupware Technologies,” *Sloan Management Review*, vol. 38, no. 2, 1997, pp. 11-21.
26. W. Orlikowski, “Learning From Notes,” *The Information Society*, vol. 9, no. , 1995, pp. 237-250.
27. D. Leonard-Barton, “Implementation Characteristics of Organizational Innovations,” *Communication Research*, vol. 15, no. 5, 1988, pp. 603-631.

28. R.G. Fichman and C.F. Kemerer, "The Illusory Diffusion of Innovation: An Examination of Assimilation Gaps," *Information Systems Research*, to appear, 1998.