USING A TEMPLATE-BASED APPROACH TO SYSTEMS DELIVERY

J. Debra Hofman
John F. Rockart

August 1993

CISR WP No. 259
Sloan WP No. 3598-93

Center for Information Systems Research
Sloan School of Management
Massachusetts Institute of Technology

USING A TEMPLATE-BASED APPROACH TO SYSTEMS DELIVERY

J. Debra Hofman and John F. Rockart

Abstract

Despite significant investments in information technology and software development, many IS organizations have difficulties delivering quality software on time and on budget. An important trend which we see emerging to address this is the use of application templates. The term "template" is being used here to describe a system -- or a portion of a system -- built with a CASE tool and reused. A template contains models of the system (data, process, and/or screen models), and customization of the template is done at the model level, using the CASE tool. In effect, a template is a flexible, CASE-based package.

This paper describes the use of a template-based approach to systems delivery in three companies. Two of the companies purchased the template externally, and one company is pursuing an internal template strategy. These companies have combined the use of the template with techniques such as prototyping, JAD, iterative development, and incremental delivery. Benefits cited include lower cost, less time, improved maintainability, better fit, increased user involvement, and an improved IS-user relationship. Effective use of a template approach has major implications for the systems development process and the IS organization, and encompasses behavioral and cultural change.

I. THE SYSTEMS DEVELOPMENT PROBLEM

According to recent surveys, the average IS budget for U.S. corporations represents 2.5% of total firm revenue, with close to half spent on software. This includes only the recognized, centralized portion of the budget; according to some estimates, adding in decentralized IS budgets and hidden end-user expenditures brings the total cost of IS to an average 5% of revenue. By 1996, total IS costs are expected to grow to 8% of revenue (McPartlin, 1993; Flynn, 1993).

Despite the significant dollar outlays, many IS organizations have difficulties delivering quality software on time and on budget. According to some estimates, 55% of systems development projects are not completed on time or on budget, and 5% are not completed at all (Maglitta and Nykamp, 1991). Not surprisingly, a survey of senior business executives revealed that more than 50% do not believe they are getting value for the money they are spending on IS, and that effective and efficient delivery of systems is a clear concern.[1]

In the context of the current business climate, this dissatisfaction is not particularly surprising. U.S. companies today are typically facing an increasingly competitive global market, intense cost and performance pressures, downsizing, reorganizations, and massive cultural change. In this environment, systems which take too long, cost too much, and do not meet business needs when they *are* delivered are simply no longer acceptable. Given the amount of money spent on software, systems delivery is clearly a business process which requires attention.

---

[1] Based on a *ComputerWorld*/Andersen Consulting telephone survey of 203 chief executive, chief operating, and chief financial officers (Maglitta, 1993).

II. MAKE OR BUY? SOME ALTERNATIVES[2]

These issues are certainly not news to many CIOs and software developers. In an IS executive survey, "improving systems development" was ranked third in a list of the critical issues warranting attention, moving from ninth place the previous year.[3] To date, the basic choice in formulating a delivery strategy has been "make or buy": should the company buy software packages wherever possible, pursuing in-house development only where absolutely necessary? Or, should custom development be the preferred route?

For those companies whose primary delivery strategy is "make," or custom development, improvements to the process of developing systems that have been emphasized over the last decade include:

- *CASE Tools:* Computer-Aided Software Engineering (CASE) tools, which help to automate the process of developing systems, first emerged in the mid-1980s. While there have been some success stories, these tools have not proved to be the long awaited "silver bullet" in the development area for a variety of reasons, both technical and organizational.[4]

- *Object-Orientation:* A second software development innovation which has gained much attention in the press is the object-oriented approach.[5] While this approach has proven extremely valuable for some types of systems (multi-media, simulation,

---

[2] Portions of this section originally appeared in CISR Working Paper #250, "The Emerging Use of Application Templates" (Rockart and Hofman, 1992).

[3] Based on a survey of 407 IS executives, with corporate revenue ranging from $250 million to over $10 billion (CSC/Index, 1992).

[4] Most companies that have adopted CASE tools have realized that the introduction of these tools -- and the methodologies on which they are based -- represent major change for the IS organization, and that effective use of the tools is associated with a significant learning curve (Kemerer, 1991). Also, the benefits have generally been realized not in development productivity, but in quality and maintenance productivity. For discussion of the organizational issues around introduction of CASE tools, see, for example: Orlikowski, 1993; Friesen and Orlikowski, 1989; Chen and Norman, 1992.

[5] Where a traditional system is composed of programs that define procedures and use data, an object-oriented system is composed of self-contained objects, containing both procedures and data, that send messages to each other.

real time), there are very few business systems to date which have been developed using it.[6]

- *Reuse:* While code reuse has long been proposed as a major solution, and while some individual programmers do reuse code on an ad hoc basis, it has generally not been institutionalized. The reasons for this have been widely debated, and range from technical to cultural.[7]

- *Techniques:* Assorted techniques to speed up the existing process are in use in varying degrees and forms. These include, for example, prototyping, iterative development, joint application development (JAD),[8] and rapid application development (RAD). While many organizations report that these techniques have been helpful, major problems with the systems delivery process remain.

Rather than build software applications, many companies are increasingly emphasizing the "buy" side of the equation, and are turning to software application packages as a preferred solution. In purchasing a package, one is, in effect, "reusing" an entire system. And, reusing previously-developed components -- code, models, or entire systems -- should save time and money and provide improved quality.

Purchasing a package should allow an organization to deliver a system faster and cheaper than building it. However, this is often not the reality. In purchasing the software package, the organization is also purchasing the *business processes* which are embedded in it. These business processes may match those existing in the organization, but often do not. The choice is then to either modify the package to fit the organization's business processes, or modify the business processes to fit the package. Either choice is almost always more

---

[6] For discussion of the adoption of object-oriented methods, see: Fichman and Kemerer, 1993; and Fichman and Kemerer, 1992.

[7] For discussion of the issues around reuse, see, for example: Caldiera and Basili, 1991; Karimi, 1990; and Cusumano, 1987.

[8] In a joint application design session, "users and IS developers work together in a structured workshop led by a trained facilitator to complete information systems delivery tasks and activities" (e.g., requirements definition for a new system) (Davidson, 1993).

difficult, more expensive, and more time consuming than anticipated; in fact, a total installation cost of ten to twenty times the original purchase cost is not unheard of.

Moreover, the fact that the package is difficult to change does not disappear once it is installed. Similar to many internally-built systems, it remains difficult to change on an ongoing basis as well. In today's business environment, however, the flexibility to change the organization, its business processes, *and* the information systems which support those business processes has become critical to an organization's success.

Clearly, the issues around the systems delivery process are complex and do not lend themselves to simple solutions. It may well be that there is no one "silver bullet," but rather that it is some combination of the above (and possibly new) approaches which will provide the answer. In our research in this area, we see an important trend emerging: the use of application templates. The template-based approach is essentially a hybrid of many of the alternatives mentioned above, combining many of the best features of both custom development and packages in a single approach.

III. TEMPLATES: THREE EXAMPLES

The term "template" is being used here to describe a system -- or a portion of a system -- built with a CASE tool and reused. A template contains models of the system (data, process, and/or screen models), and customization of the template is done at the model level, using the CASE tool.[9]

---

[9] There are different kinds of templates. Depending on the particular CASE tool, some templates may be more comprehensive than others. Some contain all three models and generate code automatically from the models, while others contain only portions of this functionality. Most of the CASE vendors we know of, however, are moving towards this full functionality.

In some ways, a template is similar to a package: a package is a fully-working system that an organization "reuses." But, there are two important differences: most packages do not come with models and most packages are not built using a CASE tool. This means that in order to modify the package -- and most organizations do modify the packages they buy, often quite extensively -- an organization's IS personnel must understand the code (or hire a consultant who does) and must make the changes directly to that code. In contrast, because a template contains the models of the system (from which the code is written or generated), changes can be made directly to those models; and, the CASE technology with which the template is built facilitates the changes. If the particular CASE tool automatically generates code, there is an added benefit: once the changes are made to the models, the code can simply be automatically regenerated from the models. The critical point with a template is that it is the models that are customized, rather than the code.

The template market is only just beginning to emerge, but is growing quickly. In some cases, companies are selling templates directly to other companies. Some CASE tool vendors operate in a "broker" capacity, offering templates built by their customers. In still other cases, a company builds a template internally, and transplants it across its multiple divisions. Most significantly, some software companies are beginning to sell their packages as templates.[10]

The three company examples outlined below offer a more comprehensive picture of the use of templates. While the three companies' stories are each unique in some ways, they are all using some form of a template and combining that use with techniques such as prototyping and iterative development. They all cite significant cost and time savings, process improvements, and behavioral and cultural changes. In effect, use of a template has provided each company with two major benefits: (1) a business and technical model; and (2) a robust prototype. In the context of the "make vs. buy" decision, buying a template is better than building from scratch, for all the reasons that buying a package is such an

[10] See, for example: Ricciuti, 1993.

attractive option -- you *start* with a working system. A template is also better than a traditional package because it can more easily be changed. In essence, *a template is a flexible package.*

In the course of studying this trend, we have spoken with and visited both companies who are using templates and vendors who are selling them. The three companies discussed below, we believe, provide a representative sample of template users. At each site, interviews were conducted with two to six members of the implementation team, representing senior IS executives, IS developers, and, in one of the three cases, business users.

A. Canadian Airlines[11]

Headquartered in Calgary, Canadian Airlines (Canadian) was formed in the mid-1980s through a merger of independent airlines. Slightly smaller than Air Canada, its major competitor, it is the world's 19th largest airline, with approximately 16,000 employees and almost $3 billion in revenues. To support its newly-formed business strategy, Canadian developed an IT strategy which included, among other things, the decision to use the Information Engineering methodology and Texas Instruments' CASE tool, IEF. One of the first systems targeted for re-construction was the frequent flyer system, a highly visible and mission-critical system. Transaction volumes had long exceeded the capabilities of the existing system, which was inflexible and required constant and extensive maintenance. More importantly, the system could not keep up with the speed with which the business changed, since each new frequent flyer promotion required extensive changes to the code.

---

[11] This case also appears in CISR Working Paper #250, "The Emerging Use of Application Templates" (Rockart and Hofman, 1992).

The first step was a three-month definition of the requirements of the new system, using joint application design (JAD) sessions. Once Canadian knew what they needed, they solicited bids for development of the system. The twelve proposals they received in return ran the full gamut in terms of both cost and time to complete.

Canadian decided to purchase TWA's frequent flyer system, built using the IEF CASE tool. While it was not the lowest cost option, they felt it could offer the best value in terms of demonstrated quality and time to deliver. The purchase price included ten days of on-site support from TWA and ten days of customer support from TI for the tool. What exactly did they receive? They received a handful of floppy diskettes that contained a conceptual model of the business process, and the resulting system design. Specifically, they received the data models, process models, screens, and code. They received no binders or documents; the documentation was on the diskettes. While the code was included with the system (it was a fully working system), they used it as a device solely to ensure that they had in fact received the entire system; after the initial run, they never used it again.

Canadian then went to work enhancing and customizing the system to its requirements,[12] with seven IS people and three users. The users were trained in key aspects of the CASE tool and methodology. Changes were made to the models, and the code was regenerated by the tool. The on-site support offered by TWA was originally contracted to cover any issues that Canadian might have in understanding the functionality and/or business rules embedded in the system. Of the ten days of TWA support, Canadian used only one; they were able to easily understand the business functionality of the system through the template.

The development team completed the system within the ten months they had promised to management, despite what could have been a major snag in the seventh month.

---

[12] The changes they made were fairly extensive including, among other things, adding bilingual capabilities.

At that point, senior management made a business decision that required major changes in the very structure of the system. The frequent flyer program -- and system -- had, in the past, been separate from the lounge program and system.[13] The customer qualified for each program separately, carried separate cards, and changed privilege levels in each independently of the other. Canadian realized that while this may have made sense from an operational standpoint, it was inconvenient and complicated from the customers' perspective. In month seven of development, the decision was made to go to one card, and therefore to one system. The implications were enormous: all the business rules for qualifying, measuring, and changing privilege levels changed dramatically, and therefore the processes and data in the systems changed as well. According to Canadian, a conservative estimate for how long this change would have taken in a traditional system was six to nine months. They were able to do it in one month, and to deliver the new, enhanced system in the ten-month time frame they had promised for the frequent flyer system alone.

Canadian: Benefits

As mentioned earlier, Canadian essentially bought a business and technical model and a prototype. The business model provided the business rules, and the technical model provided the technical design approach. As Canadian explained, in the past when you bought a package "you were always buying [a business model], but I don't think you were aware of it. You thought you were buying the code. [When you buy a CASE-based] model from another company, you're very aware that what you're buying is their business area analysis." More importantly, in buying another company's business rules, Canadian found better ways of doing business, ways they had not previously considered. In addition to this business information, Canadian acquired technical expertise as well. Through the template, they bought a system design that was easier to understand than if it were in a traditional package, and was infinitely better than any other they had seen. For example, in the TWA template system, the rules for frequent flyer promotions were separated from the body of

[13] The lounge program allows members to use Canadian's airport lounges.

the system. This streamlined design enabled users to implement new frequent flyer promotions themselves, rather than requiring IS to make the changes for them. (According to Canadian, new promotions sometimes take place weekly.)

At the same time, Canadian bought a working prototype. Instead of starting off with the results of a requirements definition in three-ring binders, they started off with a working system which the users could "see...touch...and feel." Seeing a system that actually worked and needed only to be adjusted to Canadian's requirements, it was "not as great a leap of faith," as it had been in the past, for user personnel to believe that the system could be delivered in the time frame promised. Furthermore, because the system was built in a CASE tool and involved only the customization of the business rules and data to be used (with little or no coding involved), the users could then immediately sit down and work with the *business* model and make the necessary changes jointly with the IS team. That is, as Canadian systems people note, this was not a case of "building a prototype first and then building the system...[this was a case of] developing the system using a prototyping iterative approach." Because the burden of writing code was eliminated, the developers were not averse to continual iteration. And, because the users could see that changing the system was easier and faster than it had been in their past experience, they were more inclined to work with the developers.

It is important to note also that those factors that facilitate customization of the system upon initial implementation also facilitate ongoing customization over time -- or "maintenance." According to Canadian, they have two categories of maintenance: support and enhancements.[14] With the new system, support has been dramatically reduced and enhancements are significantly easier to implement, both because of a streamlined, more modular design and because the system resides in a CASE tool.[15] There is direct business

---

[14] Support and enhancements include: (1) "fixing it when it breaks;" (2) changing the system in order to implement new frequent flyer promotions; and (3) changing the system to reflect regulatory changes.

[15] Canadian has allocated 1.5 maintenance personnel to this application; this compares to 7 individuals on a system comparable in size, but residing in a different technology.

leverage to be gained from the ability to enrich the system. According to Canadian, the *business* units are now leading the way in enhancing the system because their perception is that it can be done in a reasonable time frame, at a reasonable cost and *is therefore worth the investment.*

B. PubCo

PubCo (not its real name) is a mid-sized publishing company headquartered in the Northeast, with subsidiaries worldwide in the U.K., Canada, Australia, and Singapore. As a publisher of technical and scientific books, PubCo's product structure is relatively uniform; while there are local market variations, there is significant interplay among the markets, with products from one sold in others.

Recognizing the need for a global delivery mechanism, and the leverage which could be gained from it, senior management decided in 1988 to adopt a common hardware and software platform. They chose the AS400 for the hardware, and are currently in transition from a "collage" of mainframe and mid-range computers to the AS400. To help them with systems development, they chose the Synon CASE tool.

PubCo targeted a set of core business processes with which to begin the move towards common application software worldwide.[16] Included in the initial template, referred to as the core system, were order processing, distribution, warehousing, publishing support, and fulfillment activities. Their first step was the development of a worldwide data

[16] There are three major categories of business activity in publishing. The first is book project management, which includes all the activity through the point at which a book is in the warehouse, starting with signing an author to write the book. The second major category is the core process which covers all activity from the warehouse out to the customer, and the third is sales and marketing.

model,[17] designed to reflect the needs of multiple country constituencies. As they explained it, while there is some variation among countries, "a book is a book is a book."

With the data model as the base, they developed the system first in Singapore, their smallest office.[18] The project, including learning the Synon CASE tool,[19] was completed in six months with seven developers. They then took the core system developed in Singapore and transplanted it as a template first to Australia, and then to the U.K. They are currently implementing it in the U.S., with Canada as the next step. The system has been tailored to each site. In customizing the system, no changes are made to the code itself; all changes are applied at the model level in the CASE tool, and the code is regenerated through the tool.[20] When they are finished in Canada, they plan to turn around and go back the other way, applying some of the changes that have been made further down the line to those countries in which the system has already been installed.

PubCo is also beginning work on two other systems. One subsystem of their book project management process was developed in the U.K., and is being implemented as a template in the U.S. Another book project management subsystem, as well as the sales and marketing database, are currently being developed in the U.S. and will be implemented as templates in the other countries.

---

[17] The data model was developed over a three to four month period, along with the initial prototype.

[18] While the Singapore system was easier in the sense that it was smaller, it was also more complex functionally than some of the other sites (multi-currency, etc.). Also, because there were no MIS people on-site in Singapore to provide support once the development team was gone, they knew that the system had to be completely foolproof.

[19] To speed up the learning process, they hired a developer who had significant expertise in this particular tool.

[20] This applies to the on-line portion of the system only; it does not apply to the batch portion of the system. However, the batch portion of the system is relatively minor, representing 20% of the system code.

PubCo: Internal Template Development

PubCo has developed its own systems development methodology, combining some aspects of traditional systems development with rapid application development, prototyping, and templates.

In each implementation of the core system, the basic business requirements specific to that site are added to the core functional requirements.[21] Using the core functional requirements as a base, "scripts" are completed for each of the functions or business events[22] in the system. These describe what the business event is, the flow of the screens which support it, and its data elements and reports. Using these scripts as well as the actual working system, IS and users work together to confirm the scope of the new system and identify any additional data which might be needed. Development of the new system is then accomplished by changing and adding to the existing system. Small segments of the system are presented to the users for verification as they are completed, with IS and users sitting together and making changes. If the change requires modification in the underlying program, the users will see the result the next day; if it is simply a screen change (e.g., changing a field on the screen or the flow of screens), they see the change immediately. In effect, PubCo uses the working system as a prototype, adding and changing functionality.

The existing system also serves as a model for the new system. Rather than traditional requirements *gathering*, in which a multitude of users are interviewed and detailed requirements are documented on paper, PubCo's first step utilizing the template (after the initial evaluation) is requirements *verification*. With the boundaries clearly

---

[21] While the data are very similar across sites, business processes do vary. For example, payment cycles vary by country. A normal accounts receivable cycle in the U.S. might be 30 or 90 days; in the Far East, it might be 210 days before an initial payment is expected.

[22] A "business event" might be, for example, "customer places order," which would include the following steps: answer phone, identify customer, identify product, place order, quote price, hang up.

defined by the existing system, IS and users work together to verify the functionality of the new system and to confirm its scope.

The U.S. implementation of the core template is currently underway and provides some interesting insights. There was initial resistance to the concept on the part of both the business and technical (IS) communities in the U.S. As PubCo explained, the reaction from the business side was "...[we] don't want anything to do with it...they're much smaller than us. It'll never handle our volume." A trip was organized to the U.K. for eight users to see the system, including representatives of the various user departments from the manager level down. After a week-long review of the system in which they could use the screens and see the ease with which modifications (which previously required programmer involvement) could be made, they decided to go with the core template. The system was demonstrated to them by their business peers, not by IS. As PubCo described it, "the main thing with this system is that we're trying to get MIS out of it. It is a user system. Even though they're seeing the flow of screens, they're tailoring them for their requirements. It's *their* system." One business manager who initially resisted the template system has become so convinced of its superiority, he is now marketing it to his business peers in another site.

The initial reaction from IS in the U.S. was similarly skeptical. First, they did not believe that the new hardware would be able to handle the volumes that they needed and had been able to handle on the mainframe. The second area of resistance, and the one which is more difficult to overcome, had to do with the shift from traditional systems development to template-based systems development. The current reaction is mixed; some IS people are moving quickly and embracing the new technology and methodology, while others are unable, or unwilling, to make the transition.

PubCo: Benefits

PubCo personnel believe that this approach to systems delivery provides them with significant leverage, allowing them to share -- or "reuse" -- best practices (both business and

IS), applications, knowledge, and expertise. This has some major benefits. First, this approach has allowed them to accomplish two goals simultaneously which had previously been in conflict: they can aggregate data at the firm level while tailoring the business process and system to local needs. Second, smaller sites get greater delivered functionality than they would be able to afford on their own.

Third, they believe they have been able to reduce both time (of development and maintenance) as well as cost. Comparing the use of the template to custom development, PubCo estimates savings of approximately 30%, assuming moderate modification.[23] They also estimate that purchase and customization of an external package would cost significantly more than customization of their internal template solution. Moreover, they note that they are able to avoid the difficulties of integrating an external package with their internal systems.

PubCo also believes that their internal template approach has significantly improved the quantity and quality of IS-user interaction. The ability to interact and work with the proposed system allows a deeper level of understanding for users than would be the case with a documented (textual) description, and also means that they have more faith that the system will actually be delivered. Delivering a constant stream of system segments reinforces this improved confidence. The fact that IS and the users sit together in front of the screen to make changes and the fact that those changes happen with an immediacy previously unseen provides two important benefits: (1) it contributes to the improved level of trust and interaction; and, (2) it enables the users to develop a sense of ownership which is crucial to the success of an implementation. And, while users are learning more about what it takes to deliver a system, IS is becoming more business literate as well. For IS, this ability to better understand the business is a major benefit of the template approach. As

---

[23] They must still spend some time understanding, reconfirming, retesting, and re-documenting the function. It should be noted, however, that the effort involved in understanding the function is significantly reduced by working at the model level, since it is easier to understand a model than it is to understand someone else's code.

PubCo IS people describe it, "It's the only way they'll ever get any faith in us really -- if we can prove to them that we understand what they are talking about."

According to PubCo, using the existing system as a model for a new system provides some important learning benefits for IS as well. First, PubCo has found that the system provides a useful way of learning the CASE tool. Second, they believe it is easier to understand what is in the system using the models than trying to understand the code.

An important point to highlight is the fact that the U.S. business users interacted with their *business* peers in the U.K. to make the initial template decision, rather than being convinced by IS as is more typical. PubCo believes that the template itself and the ease with which it could be modified helped to enable this interaction.

Finally, starting off with a defined scope was seen as a major benefit. That is, using the existing system as the starting point essentially provides a clearly defined boundary for the functionality of the proposed system. At the same time, however, they did not feel that this pre-defined boundary represented a constraint, since the system is relatively flexible and easy to change.

C. Western Resources

The product of a merger of several gas and electric utilities in the mid-1980s, Western Resources in Topeka, Kansas is the fifth largest combination electric and gas utility in the U.S., serving approximately 1.5 million customers. With a background in power plant construction management, Ken Wymore came on board as the new CIO in 1986. Within the first few months in his new role, Wymore realized that the company needed a new, flexible customer processing system which would take them into the next decade.

For a utility, the customer processing system is critical. As Wymore put it, "The customer system really is the nucleus of the company." It includes billing, credit and collection, meter history, transformer history, and general customer service information. Indeed, the monthly billing envelope is considered the primary "communication link" between the company and its customers. Flexibility of the new system was considered a key dimension. In addition to a rapidly changing business environment and the promise of continued merger activity, the utility industry faces constant change in its regulatory requirements. To support this need for flexibility, Wymore decided on a relational database (DB2) as the cornerstone technological requirement for the new system.

Western then developed an RFP and examined a range of potential solutions, including off-the-shelf package vendors, custom software vendors, and other utilities. There were seven people on the evaluation team: five users and two from IS. The evaluation emphasized functional fit[24] as a first priority, with technical fit second. The package solutions which existed on the market at the time were eliminated for a number of reasons. First, they were not based on DB2 and the full cost of implementation -- purchase price plus modification -- was estimated to be comparable to a custom solution. Second, Western believed that any package solution that they could acquire would be outdated in three to five years. On the other hand, a custom solution -- especially one based in what was then a new, untested technology for which they had no in-house expertise -- was a riskier choice. And, based on comparable systems at other utilities, they estimated that a custom system would take four to five years.

Western decided on a solution which they felt provided many of the benefits of both a package and a custom system: Andersen's Customer/1 DesignWare product. At the time, this was a system which was under development at another utility. In essence, what Western bought was a system design, the CASE tool with which it was built, and consultants who had industry expertise. Specifically, they bought twelve books, that included the database design

[24] The basic functional requirements of the new system had been documented in an earlier study.

and layout, screen and report layouts, and some functional decompositions (screen logic); prototyped screens; some pre-coded program shells, included as part of the CASE tool; and, of course, the business rules or processes which are embedded in the system. They also had input available from the original utility to explain any aspect of the design necessary. There was no batch system and no code for the on-line system.

They then proceeded to customize the design and complete the system. The first six months of the three year project were spent understanding exactly what was in the product they had just bought, what they still needed to develop, and in getting up to speed on the methodology, the tool, and DB2. In addition to customizing the design (they estimate that they changed it approximately 10-15%), they added all of the on-line code, all of the batch system, and four subsystems. The project peaked at 104 team members, with a mix of 60% Western and 40% Andersen personnel. Many of the changes that Western made have become part of the Customer/1 DesignWare now sold by Andersen.

Top management played a significant and visible role in the implementation in the form of an upper management steering committee comprised of the CIO, CFO, and COO. This steering committee set a clear mandate from the beginning of the project: there would be no modification of the design guide "without *good* reason."

There was also major user involvement throughout the development of the system. The nucleus of five users who were on the original evaluation team grew to a team of twenty-one full time users on the development of the system. The users learned the CASE tool and redesigned the screens themselves. In addition to some initial prototypes, the development team presented each part of the system to the users as key functional segments were completed. In these presentations, IS and users would work at the terminal with the screens themselves, making changes where necessary. Users also provided the same type of input to development of the testing and training efforts. In addition to the obvious systems development benefits from this input, there was a secondary benefit: with their

input appreciated and incorporated, these users returned to the field as advocates for the new system.

For the information systems organization at Western, the changes which were implemented were significant, involving new approaches, methodologies, and tools. Western lost several people with solid programming expertise who were unable to make the transition.

The ability to quickly modify the system was tested four months after its implementation, as Western merged with another gas and electric utility. Some of the changes were major, reflecting different rate structures, payment plans, and billing practices. They were able to accomplish the merger in 2½ months, including the system modification and data conversion.

Western: Benefits

Western estimates that it saved approximately 12-18 months and $20 million by using DesignWare over a custom solution. They also believe that they have achieved their target of paying for the system in less than three years through head count and other cost reductions.

However, while these time and cost savings are important, Western believes it gained other benefits that are even more significant. The system design provided both a business and technical model, serving as a learning vehicle for both IS and business personnel. On the technical side, the Customer/1 DesignWare helped the IS people learn DB2 and the new CASE tool. As one of the key developers at Western explained, the IS people knew nothing about DB2. If they had been designing from scratch, they would have based the new design on the old system and would not have effectively used the DB2 product; in effect, they would probably have "tried to DB2-ize the old system." The design guide gave

them something to start with and learn from, and they could fill in any gaps by talking to the designers at the original utility.

The DesignWare also contained business process information that provided new ideas, rather than simply automating the current business process. At the same time, however, the design helped define the scope of the project, and keep it within predefined boundaries. As Wymore explained, "...if we started from scratch...we would have undoubtedly spent far more time on the design...the scope would have gotten out of control...if you don't start out with a fence surrounding the project, it is everything in the world to everyone."

Western also believes that it gained some leverage in certain technical aspects of the system. While the CASE tool did not generate code, it did come with pre-coded "program shells" for some of the technical functions common to all programs (e.g., validation edit routines). The shells provided two benefits. First, initial development was faster because some of the work had already been done and the programmers could focus on the business rather than technical functions of the programs. Second, the program shells enforced a level of standardization which, in turn, has made maintenance easier.[25]

Finally, Western emphasized some unexpected, but significant, benefits arising from an increase in user involvement. As described, users and IS worked together on the development of the system, with users designing new screens and modifying existing ones using the CASE tool. They also helped to design and implement the training and system tests. An important outcome of this greater involvement was that IS and the users learned more about each others' jobs: users began to understand the process of systems development, while IS gained an appreciation of what goes on in the field on a daily basis.

---

[25] One difference between this company and the other two examples is in the area of maintenance. In the Canadian and PubCo examples, changes were made to models rather than code during both development and maintenance. At Western, changes were made to models during development; however, once the code was generated (i.e., during maintenance) changes were applied to the code itself.

Moreover, both groups were learning at the same time, focusing together on operational screens rather than on each other's shortcomings, as is too often the case in a traditional systems development effort. This shared learning experience provided the foundation for a greater sense of partnership, in which the development of the system was a mutual effort and a shared responsibility.

IV. DISCUSSION

While there are certainly differences among the three company examples, there are some striking similarities in what these companies are doing, how they are doing it, and in the benefits they note. The benefits are significant. All three companies noted that: (1) they were able to deliver their respective systems faster and at lower cost than had been possible in the past;[26] (2) that these systems were more maintainable (and therefore could accommodate future change); and (3) that the systems were closer to what the users wanted. The question is, why? What do these companies have in common in their approaches which may be enabling these common benefits?

In all three cases, an existing system (or portion of a system) is being used as a *model* or template, for a new system. Canadian and Western took a system developed externally by another company and applied it within their own companies. At PubCo, a system developed internally is being applied to multiple locations within the company. At first glance, this may not appear particularly noteworthy; after all, it has been common practice for years to use a software package as the starting point for a new system. But what these companies -- as well as a number of others we have seen -- are doing is very different.

---

[26] It should be noted, however, that the implementation of a new system encompasses many activities other than development of the software, including, for example, conversion, integration with existing systems, training/education, and integration with new or existing organizations and processes. As such, even if development time could be reduced to zero, *implementation* time would not.

All three companies are, in effect, *reusing models*. They are using the models of the system (data, process, and/or screen) to *understand and learn* what is in the existing system; and, the necessary *changes* are being made to these models, rather than to the code itself. In effect, the work is being done at higher levels of abstraction than is typical in a traditional systems development effort. These companies are working with design-level models rather than trying to understand and change the code, in all its overwhelming detail.

But the template-based approach described here goes beyond the purely technical realm, extending into the systems delivery process itself. There are some key underlying similarities in *the way these companies used* the templates, or models. Consider the contrast between a traditional systems delivery project, and the approach taken by these companies. Most traditional systems delivery efforts generally do the following: first, a project is defined -- "we need a new order processing system." The next step is for IS to define and document the requirements of the new system in as much detail as possible, usually by interviewing as many users as possible to "extract" information about the business process. Once the requirements are fully documented, they are presented to the users for "signoff," at which point they are "frozen." IS then designs, develops, tests, and implements the system, typically with minimal user input or involvement. Alternatively, the project team might stop after the requirements definition phase and look for a package which matches the detailed requirements as closely as possible. They would then attempt some combination of changing the package to fit the requirements, and changing the organization to fit the package.

In contrast, our three company examples *used the models to help define their requirements*. Rather than starting the project with a detailed list of requirements, these companies began with a set of high-level functional requirements. The template, which satisfied these high-level requirements, provided the basic structure of the new system. IS and business personnel then *jointly* carved out the details of the new system through an iterative process of working directly with the screens and changing the template. Rather

than two sequential steps -- first defining all the requirements and then making all the necessary changes -- they arrived at a new system through an iterative and interwoven process of requirements definition and system modification, delivering key functional segments of the system as they were completed.

The companies all noted certain benefits in the template process described above. In all three of our company examples, the template served as a learning vehicle for both IS and business personnel. For IS, the template provided a useful introduction to the new tools and systems approaches they needed: the CASE tool, some technologies (e.g., DB2) and a model technical design. On the business side, the template provided new ideas and business processes. However, templates provide more than simple knowledge transfer; after all, external ideas, knowledge, and expertise can be transferred to an organization from traditional types of packages as well. More importantly, templates provide the capability for both IS and users to jointly *interact* with the new system -- to understand and make changes easily. It is this joint, hands-on interaction which primarily distinguishes templates from packages. And, it is this interaction which facilitates learning.

In addition to learning more about their own functions, IS and users learned about each other's as well -- and, they *learned together*. Thus, the template served not only as a learning vehicle, but as a *communication* vehicle. In effect, it provided a forum in which IS and users could communicate with each other, understand each other's jobs better, and begin to build a partnership and a sense of mutual responsibility for the delivery of the system. Using the screens to visualize and articulate the requirements of the new system, rather than a blank sheet of paper, helped not only to improve the IS-user relationship, but also to improve the system itself.

In all three cases, a sense of user *ownership*, not simply involvement, was articulated. The users' experiences with the system -- the ability to interact with, communicate through, and make changes directly to the system -- helped to foster this.

The template process described here, then, is a process characterized by improved learning, improved IS-user interaction, and user ownership. There are, however, some other elements of this process which offer important benefits as well.

- By starting a project with only the basic requirements rather than with voluminous detail, these companies were more inclined to reuse a system or portion of a system which already existed, rather than automatically taking the "full custom development" route. Several of our interviewees noted that starting with a detailed list of requirements gathered from everyone involved will almost always result in a custom developed system or a heavily modified package, since no existing system could precisely match the several thousand requirements that are typically gathered with this approach.

- Beginning the detailed requirements phase with the template helped to contain the scope -- and therefore time, cost, and expectations -- of the project.

- The incremental nature of the delivery of the system -- the fact that users see results more quickly -- also helped improve the credibility of IS and the level of trust.

- The template process described here allows the requirements of the new system to evolve, better accommodating the reality of business change.

The template-based approach described here is multi-faceted, encompassing both technical and process changes. The potential benefits are noteworthy: lower cost, less time, more flexibility, better fit, improved IS-user relationship, increased user involvement, user ownership and control, external knowledge, greater customization ability, and an improved learning capability. For a company pursuing an "internal templates" strategy there can be additional benefits: the ability to leverage best practices across the organization, and to share applications, knowledge, and expertise.

Some of the benefits noted are possible with the purchase of an off-the-shelf software package. Others are possible with a custom solution, particularly one which uses techniques such as prototyping, joint application development (JAD), evolutionary or adaptive design and development, CASE, and reuse. *This* solution -- the template-based approach -- offers

many of the benefits of both. According to these companies, it provides many of the benefits typically expected of a package purchase: compared to custom development, it takes less time and costs less, external expertise and ideas are provided, and scope is defined at the start. At the same time, it provides many of the benefits companies often seek in pursuing a custom development route: compared to a package purchase, the template is easier to understand and customize, satisfies more of their requirements, and can provide the user involvement and sense of ownership that a typical package purchase does not allow. As one of our interviewees put it, "this allows the user to live the building of that system" and to develop a sense that it is *their* system.

V. Issues and Implications

The use of the template-based approach described here has some important organizational implications. Alluded to throughout the case descriptions and discussion, some of these implications are obvious; others are, perhaps, more subtle.

The project team for a template-based systems delivery project is different than for traditional methods in terms of composition, structure, roles, responsibilities, skills, and mindset. In general, the template teams tended to be smaller, with more users and fewer IS people than traditional teams. Roles and responsibilities for each member of the team tended to be more diverse, with a greater need for business-oriented, communication, and interpersonal skills.

The template-based approach also requires some changes in system design and development skills and mindset. A company pursuing an "internal templates" delivery strategy is, in effect, "reusing" a system from one site to another. In order for this to be possible, the system must be *designed* for reuse; in the case of PubCo, for example, the initial worldwide data model had to be designed in a way that would reflect the needs of

the multiple countries.[27] This requires different problem-solving skills than those used in traditional system design.

Change is also required in the *development* effort, particularly with regard to behavioral and cultural issues for both IS and users. This approach requires a shift from the "not invented here" mindset which seems to be typical of many IS development groups. Business users also tend to believe that their business process is unique and fundamentally different than others, and that, therefore, a previously developed system cannot be used without major modifications. In fact, there is more commonality than is typically understood or admitted.

The use of the template as a communication vehicle suggests the possibility of an interesting and fundamental shift in the nature of the systems delivery process. Using the templates (screens, in particular) to define and delineate a new system, rather than arcane IS modeling tools, means that the language of the discussion between the business users and IS can shift from a heavy use of IS technical jargon to the language of the business itself. In addition to the factors discussed in the previous section, this contributes to the increased sense of user ownership and control.[28]

There are some potential issues associated with the use of templates. One of the benefits cited by these companies was that the template provided a defined scope at the start of the project. The question which arises is, is it possible that starting with a pre-defined scope might be a constraint, potentially limiting creative new ideas and solutions?

---

[27] Designing a system for reuse involves both conceptual and technical aspects. Conceptually, the functions to be performed by the system must be analyzed for their underlying similarities. (On the surface, this may appear to be a relatively straightforward task; in fact, it is one which is highly complex.) Given these similarities, the functions must then be defined generically so as to apply to as many instances as possible. There are also some technical aspects to designing a system to be reused. Briefly, it should be modular, streamlined, and engineered to leverage the commonalities across its composite functions. For a discussion of this see, for example, Hess, 1990.

[28] Of course, it is important to recognize that this shift in power and control may not be perceived by all members of an organization as positive, and that this perception must be managed carefully.

It is possible. The safeguard, however, should come before the template is selected. With the template-based approach, the step prior to deciding upon the template is one in which the high-level requirements of the new system are defined. It is during this step that creativity should be unconstrained.

There may also be some situations, especially at this point in time when the template market is new and supply is somewhat limited, in which it is more appropriate to buy a traditional package. Western believes, for example, that while a template was appropriate for a system of this size, complexity, and criticality, a package solution would be more economically appropriate for certain other types of systems which require less customization (e.g., general ledger). Another organization which did install a general ledger template disagreed with this, however, citing all the advantages of templates previously discussed.

Finally, there are some issues around the use of CASE. A template is a system built in a CASE tool. In order to take full advantage of this fact, a company that purchases a template and wants to customize it should be making changes to the models, not to the code itself. This means that the company must have -- and know how to use -- not only the template but also the underlying CASE tool. For some companies, this can be a very useful means by which to introduce the tool. For other companies it may not be as useful, particularly if a company has already chosen another tool.[29] In any case, the decision to purchase a template implies a decision regarding CASE which should be made in the context of the specific business and development environment. This can have major implications (Rockart and Hofman, 1992).

Clearly, these changes in skills, roles, and responsibilities -- resulting both from the use of the template-based approach as well as the use of the CASE tools with which the templates are built and maintained -- must be managed, and imply a need for training and

[29] That is, the goal is not to have multiple CASE tools which are redundant in functionality, or tools which do not overlap in functionality but which also are not connected to each other. This would only increase complexity. This will become less of an issue when and if cross-CASE bridges become available.

education. The cost of this, both in terms of time and money, is likely to be high. And, for many companies, a successful change will require changes in reward and incentive systems.

Perhaps most importantly, the type of change described here requires leadership and active involvement on the part of management in order to be successful. For example, at Western, senior management set forth a clear mandate of "no change without good reason." At PubCo, the driving factor in the pursuit of templates as a key component of the systems delivery strategy was the belief on the part of senior management that there was commonality in business processes across organizational divisions, and that this strategy would allow them to leverage this commonality to be more competitive.

It is clear, from many of our discussions, that there is major interest in and potential demand for templates. On the supply side, the template market continues to evolve. In addition to the CASE tool vendors and software suppliers mentioned previously, vendors of templates are beginning to emerge, and the software market is undergoing transformation.

REFERENCES

Caldiera, G. and Basili, V., "Identifying and Qualifying Reusable Software Components," *IEEE Computer*, February 1991, pp. 61-70.

Chen, M. and Norman, R., "Integrated Computer-Aided Software Engineering (CASE): Adoption, Implementation, and Impacts," *IEEE*, 0073-1129, 1992, pp. 362-372.

CSC/Index, "Critical Issues of Information Systems Management for 1993," *The Sixth Annual Survey of IS Management Issues*, 1992.

Cusumano, M., "The 'Factory' Approach to Large-Scale Software Development: Implications for Strategy, Technology, and Structure," MIT Sloan School of Management Working Paper #1885-87, September 1987.

Davidson, L., "An Exploratory Study of Joint Application Design (JAD) in Information Systems Delivery," MIT Sloan School of Management, Center for Information Systems Research, Working Paper No. 258, Cambridge, MA, June 1993.

Fichman, R. and Kemerer, C., "Adoption of Software Engineering Process Innovations: The Case of Object Orientation," *Sloan Management Review*, Vol. 34, No. 2, Winter 1993, pp. 7-22.

Fichman, R. and Kemerer, C., "Object-oriented and Conventional Analysis and Design Methodologies: Comparison and Critique," *IEEE Computer*, Vol. 25, No. 10, October 1992, pp. 22-39.

Flynn, S., "Information Technology Budgets," Gartner Group, 1993, Conference Presentation.

Friesen, M. and Orlikowski, W., "Assimilating Case Tools in Organizations: An Empirical Study of the Process and Context of CASE Tools," MIT Sloan School of Management, Center for Information Systems Research, Working Paper No. 199, Cambridge, MA, October, 1989.

Hess, M., "Information Systems Design in Industrial Practice," in *Concise Encyclopedia of Information Processing in Systems and Organizations*, ed. A.P. Sage, Pergamon Press, May 1990, pp. 1-12.

Karimi, J., "An Asset-based Systems Development Approach to Software Reusability," *MIS Quarterly*, Vol. 14, No. 2, June 1990, pp. 179-198.

Kemerer, C., "Learning Curve Models for Integrated CASE Tool Management," MIT Sloan School of Management, Center for Information Systems Research, Working Paper No. 231, Cambridge, MA, November, 1991.

Maglitta, J., "Squeeze Play," *ComputerWorld*, April 19, 1993, pp. 86-91.

Maglitta, J. and Nykamp, S., "Software Speeder-uppers," *ComputerWorld*, August 26, 1991, pp. 51-53.

McPartlin, J., "Not the Best of Times," *Information Week*, June 21, 1993, p. 74.

Orlikowski, W., "CASE Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development," MIT Sloan School of Management, Center for Information Systems Research, Working Paper No. 255, Cambridge, MA, May, 1993, forthcoming in *MIS Quarterly*.

Ricciuti, M., "Build Custom Apps at Packaged Prices," *Datamation*, June 1, 1993, pp. 71-72.

Rockart, J. and Hofman, J., "The Emerging Use of Application Templates," MIT Sloan School of Management, Center for Information Systems Research, Working Paper No. 250, Cambridge, MA, December 1992.