Introducing shareable frameworks into a procedural development environment

by R. L. van der Salm

Typically, an IBM midrange-oriented independent software vendor such as the Dutch company Consist B.V. is not a leader in using new technology. Such companies prefer to use stable, proven technology environments to create and sell their products rather than take the risk of using new, unproven technologies. Consist B.V. has been known for over 20 years in the Benelux market for delivering stable applications and investing only in proven technology. Why is it, then, that they now invest large sums of money in new technologies such as framework-based development, Java™, CORBA™, and object-oriented programming? This paper describes the business reasons that influenced Consist B.V. to make this shift in its strategy and the way in which the traditional procedurally oriented development organization is changed into an object-oriented, framework-using software factory. Apart from the technical implications and the steep learning curve associated with using these technologies, some attention is also given to the human resource management aspects of the organization.

If you do not create your destiny, you will have your fate inflicted upon you.

—William Irwin Thompson

Vendors of standard software applications (independent software vendors, or ISVs) worldwide find it more and more difficult to invest in modernizing their applications, for several reasons. In the past, most standard applications took several years to grow from being very simple and small, to be used by one or just a few customers, to becoming large applications. As the size and complexity of the applications gradually increased, the investment accompanying the increase could be spread over several years.

Because application technology is now changing from procedural to distributed object solutions, ISVs must make an important decision: Do they invest in their current procedural applications and continue with the strategy they have followed for years, or do they invest in completely new applications with completely new technology to be able to survive in the future? Both strategies have advantages and disadvantages. The easy choice is to maintain the old applications and offer more functionality in the existing modules. The investment needed is not very high, and the short-term risk profile is low. However, the longer-term risk in this scenario is extremely high. If the demand for open systems really changes into a demand for open applications, if object-oriented technologies really become the standard of the future, if the Java** programming language really turns out to be more than "hype," if the network computer really becomes the success that is predicted, and if new technologies really keep on being created with the speed with which they have been, then applications need to be rewritten with object technology now, to keep pace with all these rapid developments.

0018-8670/98/$5.00 © 1998 IBM

Deciding to rewrite existing applications so that they are based on new technologies is not an easy choice. The investment needed to create completely new applications that have at least the same functionality as the existing applications is enormous.

At this moment, choosing an object-oriented language that is multiplatform and supports all the other trends listed above means choosing Java. This choice increases the current risk because of the innovative character of Java and the lack of stable tools and standards for the Java environment. Apart from such technical risks, there is another very important risk for ISVs who choose to switch to using object technology and building completely new applications. Switching from procedural development to object-oriented development is not easy; it takes time, and not everyone will be able to do it. Some of the ISV developers who are keeping the current product lines up and running want to make the switch immediately, because they fear they will be left behind if they do not learn to use the new paradigms as soon as possible. Human resource management in the ISV development organizations has an extremely important part in keeping developers happy with what they are doing, and seeing that key people are not lost to competitors because they may think that they are no longer important.

Like other ISVs, the Dutch company Consist B.V. faced these problems and had choices to make. Consist B.V. chose a strategy based on using IBM's San Francisco* product, a shareable framework technology. This paper describes the business reasons for choosing San Francisco, the way in which the switch from procedural to object-oriented development is being made, the human resource management issues that occur with these decisions, and the first programming experiences with San Francisco. We begin with some background information about Consist.

Consist B.V.

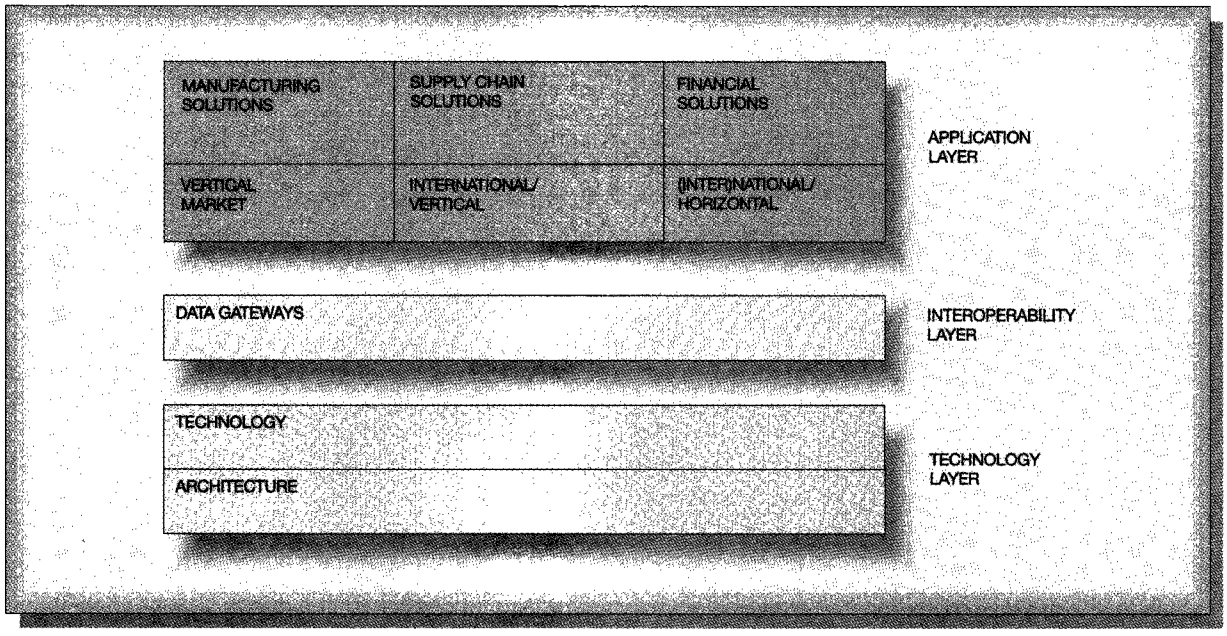
Consist B.V. has been selling financial and human resource applications in the Dutch and Belgian market for over 20 years. The environment in which it delivered several generations of its applications has always been the IBM midrange computing area. The applications it produced grew or were rebuilt on platforms such as the IBM System 3X and, since the late 1980s, on the AS/400* (Application System/400*). Although these platforms are, of course, quite different from each other, the growth from one platform technology to the next was always relatively easy and natural. All environments had RPG (Report Program Generator) as the main programming language, so Consist developers could easily adapt to the next generation of computers. For an application vendor, this is an extremely important issue. Not only were the developers able to grow easily into a new technology, but customers could do so also.

An application vendor such as Consist has two important assets. First is the customer base that is stable and large enough to make it possible to invest in product lines to keep up with the changing environments; second is the knowledge possessed by its employees. Developers who have functional knowledge of basic customer needs and wants will be in demand in the future, and those able to translate these customer requirements into technical implications are of great value to the company. It must be clear that, whenever it is possible, both the customer base and the developers must be cherished and led to the next generation of technologies in stages that are easily assimilated.

So far Consist has done a good job of this, being the market leader for financial and human resource applications aimed at middle-sized and large companies (defined as companies having over 100 employees), residing in the home market of Baan, with neighbor SAP watching closely. Consist has 1200 middle-sized and large customers based mainly in the Netherlands and Belgium. About 150 implementations are done elsewhere in Europe. Among these 1200 customers there have been about 1400 installations of Consist applications.

Although the above sounds like a very good base on which to set the next small step of the technology evolution—such as extending the client/server possibilities of its applications—two years ago the Consist management made a difficult decision. The technology strategy was shifted from using only procedural languages such as RPG and Synon's Synon2/E to using object technology. The impact of this decision was much greater than expected. Learning to use object technology is not easy for a group of experienced RPG developers. In addition, although the reuse that is possible in an object-oriented environment may sound very good to a manager, many developers have the "not invented here" syndrome, and are not willing to change their old and proven ways of working. A whole new approach to building systems and a whole new set of skills and tools are required. In addition, the investment costs of changing the way in which the software labs are now work-

Figure 1 Integral application architecture



ing and starting a new product line are very high. Before Consist made the decision to enter this new world, much research was done to investigate how its environment would change in both the short term and the longer term. After that, a plan for how to act on these environmental changes was developed.

Business reasons for choosing San Francisco

"Would you tell me, please, which way I ought to go from here?" "That depends a good deal on where you want to get to," said the Cat. "I don't much care where," said Alice. "Then it doesn't matter which way you go," said the Cat. "So long as I get SOMEWHERE," Alice added as an explanation. "Oh, you're sure to do that," said the Cat, "if only you walk long enough."

—Lewis Carroll, Alice and the Cheshire Cat

In the information technology (IT) world, there is a lot of hype for the most recent developments in computing. Among such current developments are object technology, the Internet, intranets, extranets, Java, and the network computer. An ISV such as Consist that has a large customer base can only proceed slowly with these newly developed technologies. Only if a technology is really becoming a trend, and it is clear that a trend will become a standard, can the technology be put into products. Therefore, it is extremely important to recognize developing technologies and trends, and to have an idea, or vision, of where the market will be going. Using new technologies too soon may cause severe problems for existing customers, because a technology may be unproven and unstable. Adopting new technology too late in a highly competitive market, such as the one for financial and human resource applications, may cause a competitive disadvantage. Consist now thinks that the market for ISVs is changing as follows.

Complete vs specialized solutions. At this moment there are two kinds of suppliers of standard applications. One kind includes the large vendors led by SAP, Baan, Oracle Corp., and Peoplesoft. These vendors offer their customers an integral solution for all their information needs, a practice called ERP (enterprise resource planning). These applications consist of several layers, as shown in Figure 1. The *application layer* contains the functional solutions offered by a supplier.

Although most suppliers suggest that they offer a solution that is 100 percent complete for their customers, much work almost always has to be done before

an application is fully implemented. One of the reasons for this discrepancy is that, although the large ERP software vendors try to give their customers a 100-percent solution, they are never able to support the local functionality, such as that required by the laws, rules, and habits for each country in which they are selling their products. Apart from that, each industry has its own specialized ways of working that cannot be covered in these general-use ERP applications. So, although everyone is trying very hard, a 100-percent solution can never be reached with the standard integral applications. An application can be changed through the *interoperability layer* to behave in the way a customer wants. This layer provides a set of application programming interfaces (APIs) or business application programming interfaces (BAPIs) that make it possible to interact with the data that are stored within the standard application. These APIs do not really make the application open. The technology of the application and the complete structure of the integral application remain closed. The technology that can be used for this is defined in the *technology layer*. Most large ERP application vendors have their own closed technology. An example of such a closed technology is ABAP IV from SAP.

Among the reasons that an organization decides to buy an integral ERP application is the fact that it can do business with only one supplier, and the information systems that support several processes or functions in the two organizations will all work in the same way. The coupling between these different information systems will be tight. The selection process for an integral application is based on the fit between the requirements of the organization and the functionality an application offers. For obvious reasons, when defining how close the fit should be, the fit affecting the primary processes is the most important. Therefore, a chosen integral application fits best with the primary processes; other processes of the organizations do not each utilize a best-of-breed application, but also use the chosen application.

The other kind of application vendor delivers a specialized application for a horizontal or vertical market. Examples of these kinds of applications are financial, human resource management, fixed asset, logistic, and local government solutions. These applications are not integral because they support only part of the information needs of an organization. Technically these applications are built on generally available technologies such as RPG, COBOL, Oracle, Synon2/E, and Progress. The interoperability layer, depicted in Figure 2, is very open, because it is important that the applications be able to connect to other applications. As a consequence, the integration between different applications is loose.

An organization that decides not to buy an integral application but to buy several applications and connect them through interfaces, creates the possibility that it will buy the best-of-breed application for each vertical process in the organization. In cases where a global integral ERP supplier is not able to support all the local functionality in its applications, such as that for laws and habits, this local functionality will be supported in these specialized applications. This makes the implementation much easier, because the standard application does not have to be changed very much to fit the needs of the customer. However, the interfaces between the separate applications will be loose, and the user interfaces of the separate applications will not necessarily have the same "look and feel."

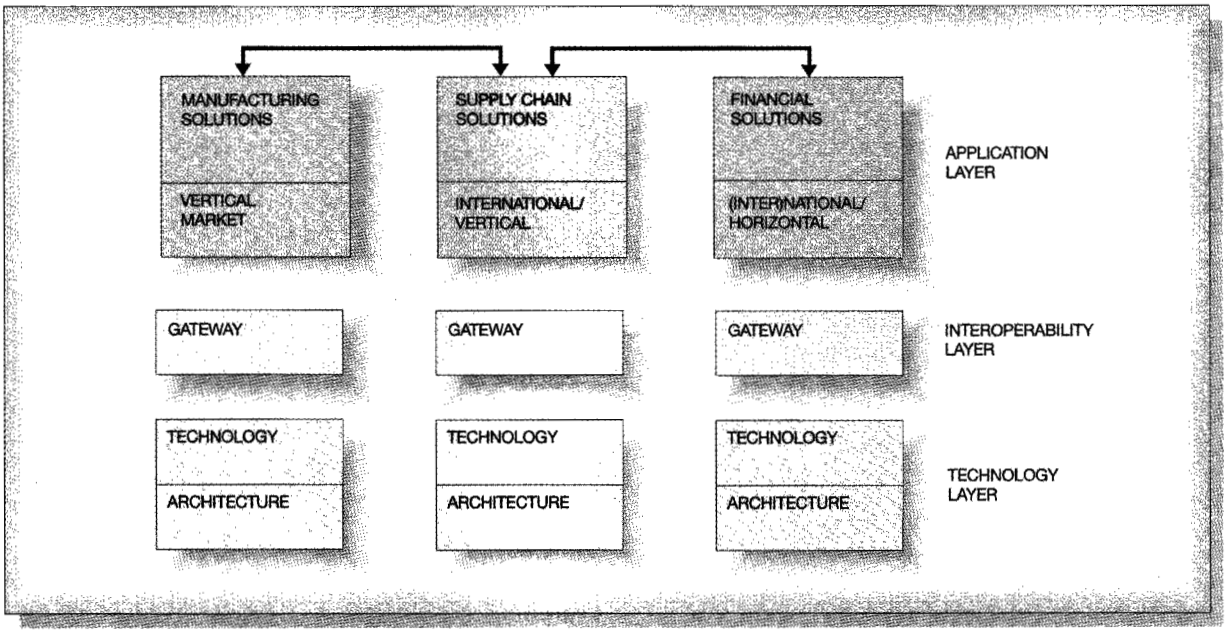
It can be seen from the above discussion that both approaches to delivering standard applications have typical advantages and disadvantages. Depending on the size and the organizational structure of the customer organization, one of the approaches will be chosen.

However, the way in which the standard application industry works will change. Application vendors have to make a choice. All vendors, even the large ones like SAP, Baan, and Oracle, have discovered that they cannot supply the market with 100-percent complete solutions. Deutsche Morgan Grenfell Technology Group describes that situation as follows: "Expanding the scope of coordination further into the enterprise and across midsize enterprises will require overcoming major technology hurdles. It means evolving the applications to include a flexible backbone that others can build on and extend as the applications' reach continues to expand. *Every company has its own unique requirements that cannot be fully anticipated by a single vendor.*"[1]

Another view of how the role of the standard application vendor will change is given by Martin Healy, an IT industry watcher who writes his own column for some IT magazines:

> There is a real problem in implementing standard applications. A standard application reduces the programming work an organization has to do but that's all. Applications also need to be imple-

Figure 2 Specialized application architecture



mented. Implementing still demands a design, knowledge, skills, training, etc. The level of integration is overexaggerated. In all cases after buying a standard application a lot of work has to be done to implement an almost always incompatible application. The real answer for this is reusable code, where a complete suite of reusable business objects is available and applications are being built by using the necessary modules. To realize this, a rigorous definition of the functionality that these objects should be doing is necessary. This development has already been started and will continue. In ten years we won't be using standard applications anymore. At that time we will develop our own applications again, but with reusable, modular technology.[2]

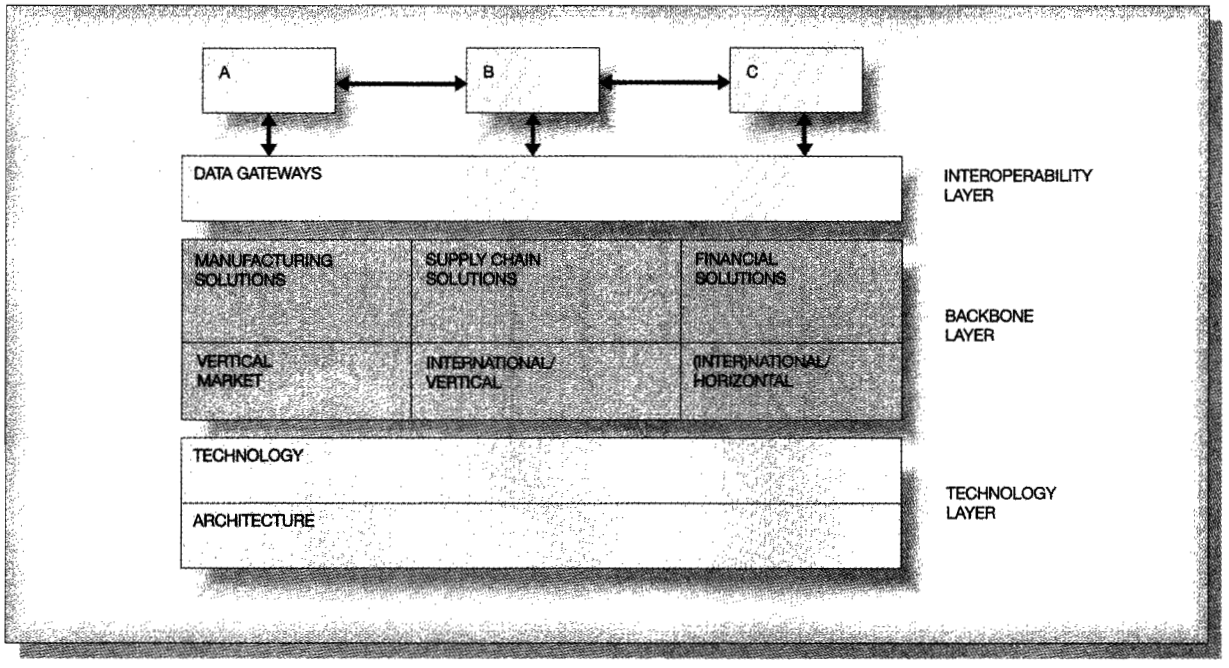
Future requirements. In the future, customers will require the following:

1. An application must be able to be tailored to the customer's unique requirements.
2. Tailoring must be done in a rapid implementation through easy-to-use technologies.
3. The integration of specialized software from other vendors must easily adapt to the industry's and customers' specific needs.

These needs can be summarized in view of the market's requirements for backbone technology. Backbones will be the base layer of tailor-made, industry-specific information systems. A main reason for tailor-made information systems is to give companies the ability to obtain a competitive advantage from their information systems. When every company in the world uses exactly the same ERP system, it will be difficult to differentiate a company from its competitors based on the use of its information resources.

The need for backbone technology has already been acknowledged by the application vendors. Although at this moment no backbones incorporating reuse are available, they are being created. The structure of the industry will therefore change. Earlier in this section, two kinds of application vendors in the current industry structure were defined: the integral ERP application vendor and the specialized application vendor. Both types of vendors will change the way in which they work. The large integral ERP application vendors will change their goals of delivering 100-percent complete solutions for their customers, to delivering a 70-percent solution that can be brought up to 100 percent through tailoring of the available backbone. This will be done by using customized soft-

Figure 3 Closed backbone technology



ware and by integrating specialized software from other vendors.

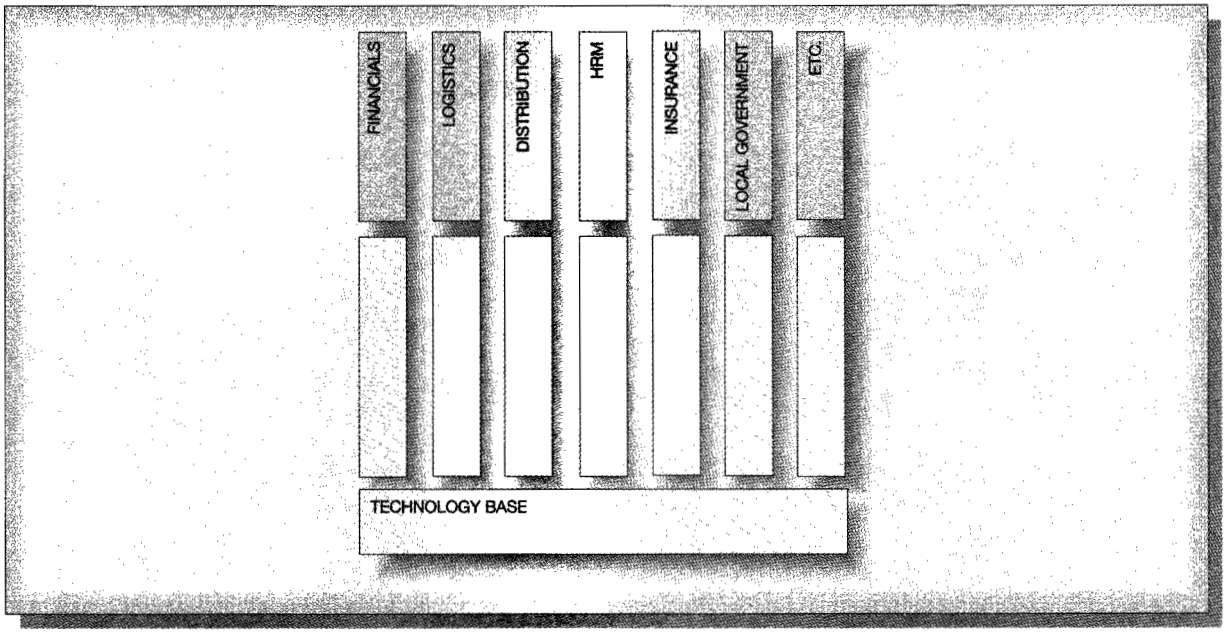
This change means that the ERP vendors will deliver their own backbone, based on their own existing technology of reusable components. Through the interoperability layer, this backbone can be tailored to the specific needs of a customer. This backbone can be defined as a closed backbone, because the technology that is used is from one vendor and is not a common one, as shown in Figure 3. Only the very large ERP application vendors are able to develop their own backbone. Specialized vendors can adapt the backbone and integrate their specialized application to the backbone of one specific backbone vendor. In doing so, these specialized vendors become dependent on one backbone vendor.

The other application vendors have to make a choice. Do they adapt to the closed backbone of one of the large vendors, or do they adapt to a more open industry standard? One of the approaches used to create an open backbone is available from the Object Management Group (OMG). If an OMG standard can be created, this open backbone will provide about 40 percent of the functionality needed for an application. This 40 percent consists of reusable components or frameworks. Application vendors can build their specialized functionality on this base.

Because of its open architecture, an open backbone gives vendors that use this architecture the possibility of having their applications interface with one another through the technology base layer illustrated in Figure 4. In turn, the interfacing between applications from different vendors can become much tighter than it is now. Therefore, customers have the chance to choose the best-of-breed application per vertical process and still have the horizontal integration that an integral application offers.

Some years ago the need for open platforms was responsible for a shift in the way the IT industry was structured and how the power in the industry was divided. Because of the Java revolution, the birth of backbone technology, and the need for mass customization of customers' standard applications, the drive for open platforms will grow into a demand for open applications in the coming years. Consist considers IBM's San Francisco to be the most promising technology to fulfill this demand.

Figure 4 Open backbone technology



Implementing a framework strategy in an RPG environment

The significant problems we face cannot be solved at the same level of thinking where we were when we created them.

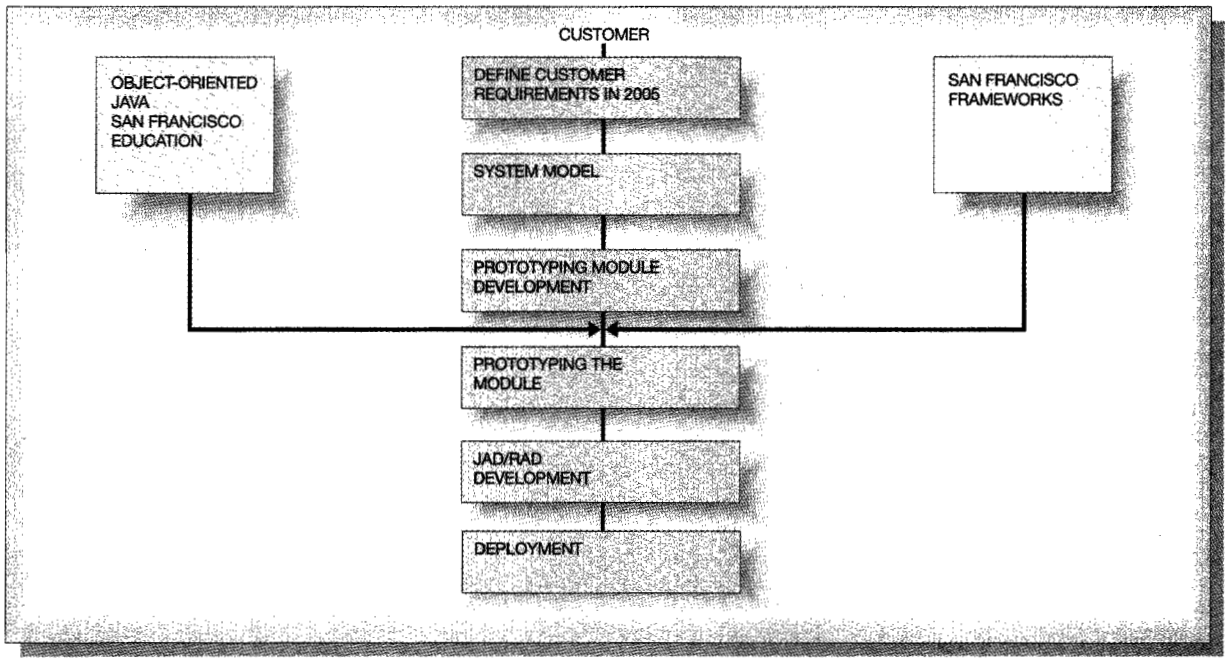
—Albert Einstein

The strategy that Consist B.V. intends to follow may be surmised from the above discussion. Consist had to choose a backbone technology, build a new product line based on that technology, and then sell it. Of course, life is not as simple as that statement. Consist has much experience in building RPG-based applications. Working in such an environment differs completely from working in an object-oriented environment. Currently, a small group of developers at Consist has object-oriented experience, but the majority is procedure-oriented. The way Consist implemented this strategy is shown in Figure 5.

Forming the shareable frameworks department. Upon concluding that the new Consist product line will be framework-based, multiplatform, and object-oriented, and, after some research based on San Francisco, Consist created a special department, the shareable framework department. Its goal is to develop a new product line for Consist, based on framework technology to make sure Consist will survive in the longer term. From the moment of its creation, 90 percent of the developers decided that they should work for the new department. The challenge of working with object technology, Java, San Francisco, etc., attracts almost every developer who likes the profession. Not being within the first group working with these new technologies may give developers the idea that they are not important to the company. Of course, this is not true. Developers who work for a standard applications vendor are a special breed. In addition to having the necessary technical skills, they also build up much functional experience. The investment needed to transform these procedural developers to object-oriented, framework-using developers is high, but the investment needed to hire new developers and transform them into application developers who have the necessary functional skills is much higher. Thus, the developers' fear of being considered unimportant over the longer term is unfounded.

The first group of developers within Consist for the shareable framework department consists of highly experienced employees. A number of top developers from other areas of the company were moved to

Figure 5 Consist B.V. San Francisco strategy implementation process



the department. Although other developers might be jealous of them, everyone can accept the fact that the top developers should go first, because of their successes in past projects. The main reason for choosing these developers is to ensure that the project is likely to succeed. The working environment in which object orientation, Java, San Francisco, CORBA** (Common Object Request Broker Architecture), workflow, etc., are used is, of course, very risky. An initial small group of good developers can create a roadmap to ease the way for later groups of developers entering the new technology environment.

The learning curve. Domain experts are needed for the functional part of such a large object-oriented project. For Consist, a domain expert is not defined as an experienced system designer or system analyst, but is someone who knows the functional domain well and has many contacts in the business and academic world. From these contacts it is possible to obtain the views of other people, including those from other disciplines, on how a functional area will change in the future. The people who are able to fulfill this role for Consist are application consultants. These employees do not have many technical skills, nor do they have any development experience. They do have much functional knowledge. Three of the most experienced consultants were selected to fill this important role. Their tasks within the project are to define the functional needs, model the use cases, write scenarios, and create the high-level class diagrams. It was a bit painful for the established departments to lose such highly experienced people to the shareable framework department. But because everyone understands that expertise gained through experience is needed to build a new product line, the situation is accepted by their colleagues.

Although it might be expected that this task would be impossible for anyone without training in designing an information system and without experience in using traditional techniques such as information engineering, data flow diagramming, and data modeling, it is not the case. Of course, some training is needed on object-oriented analysis and design, some mentoring is needed on the structure of use case documentation, and some experienced persons have to evaluate the class diagrams. However, because object-oriented modeling is much closer to the real world than procedural modeling, the learning curve for the domain experts is relatively short. The quality of the analysis work of the domain experts is very

good. The lack of design experience that may have been a big problem turns out to be an advantage, because the domain experts do not have to rid themselves of any procedural design paradigms that they might have learned in their former work. The area in which the domain experts do have much to learn is understanding what is offered by the San Francisco frameworks. The implementation and use of frameworks is technical, but San Francisco offers much functionality with many great advantages. This functionality can only be put to use when the domain experts understand what is offered. Gaining this understanding takes much time because the San Francisco documentation is technology-oriented instead of being functionally oriented.

The learning curve for domain experts to enter the object-oriented world might be steep, but is apt to be relatively easy if those selected have the proper background. The learning curve for developers is very steep and very difficult to overcome. An experienced object-oriented developer can, of course, learn Java in a relatively short period, understand the working of the San Francisco frameworks in a couple of weeks to a few months, and then start building applications. For the average AS/400 RPG programmer, the challenge is enormous. Apart from learning and understanding the object-oriented programming paradigm, learning a language like Java that differs completely from RPG is very difficult. Programmers who surmount that obstacle understand the use of design patterns but need much time to implement them. The lack of mature Java tools does not make the task any easier. After the San Francisco programming model is understood, which again takes some time, much research is required to understand what San Francisco offers as functionality and how it can be used.

Besides the learning curve for object orientation, there is a very important mind-set that developers need to have before becoming good San Francisco developers. Using frameworks instead of building everything by yourself means getting rid of the "not invented here" syndrome. A developer who does not reuse a lot of code will never be a good object-oriented developer. To implement reuse, special attention is needed from a project manager's point of view.

Structure evolution. Why is it more difficult to turn an RPG developer into a San Francisco developer than it is to turn a developer for PC programs or a developer of applications based on Oracle, for example, into a San Francisco developer? This can be explained by the evolution of the framework concept as defined by Taligent.[3] Taligent states that there are three stages in the evolution of application programming structures, as shown in Figure 6.

In the earliest stage, the procedural approach, the programmer provides all code for flow of control. The operating system has libraries with procedures that perform certain tasks that can be called. The program flow is controlled sequentially, and the system takes action only when the program calls it.

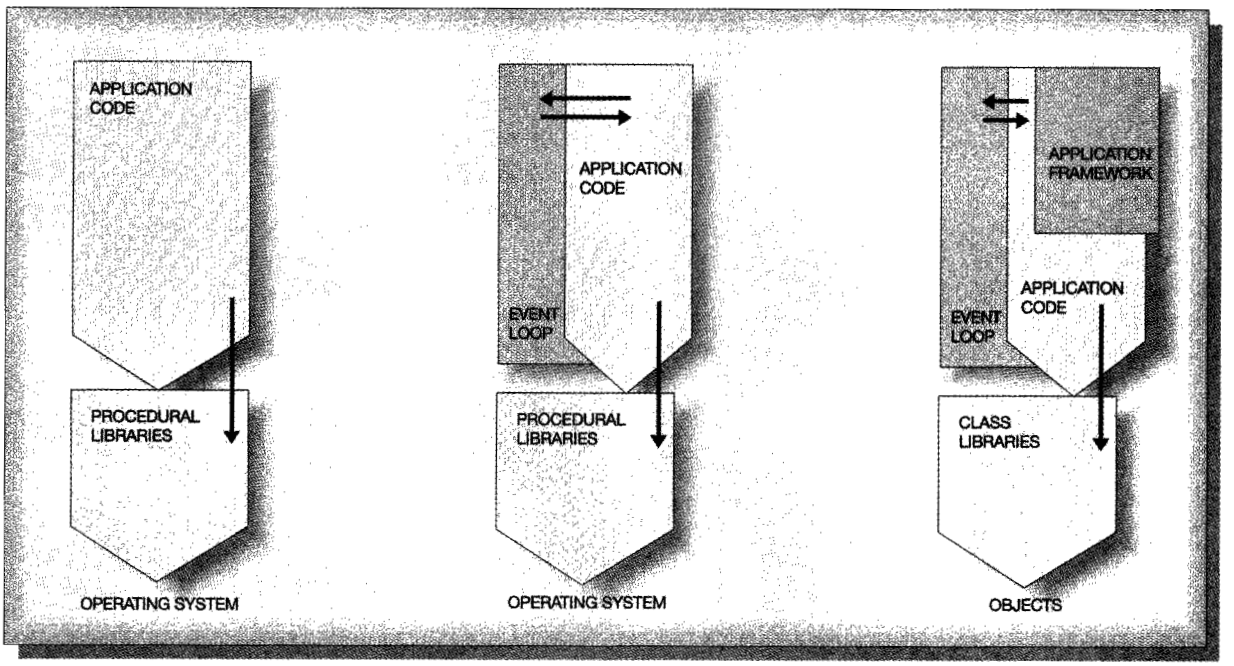
The second stage, the event loop approach, was introduced with the development of the graphical user interface. End users started to interact with applications in a different way. A sequential flow no longer could accommodate a user's choices. The user triggers an event by clicking the mouse or using a keyboard, etc. The event loop calls sections of the application program that handle the action the user requests. After the event loop call, the programmer is again responsible for the flow of control of the actions to be taken.

The third stage is the framework approach. The framework environment takes care of almost all flow of control and calls the programmer's code only when necessary.

RPG programmers typically use the procedural approach in their work. When they begin to work with the framework approach, they miss the evolutionary step of the event loop approach. Developers working with Oracle programs or writing PC applications who have never used object technology are used to giving some of the flow of control to the system. For them the step up to the framework approach is therefore smaller.

Experience has shown that it takes at least a half year before an experienced procedural developer is completely comfortable with the object-oriented, framework method of developing applications.[4] Then a developer is able to create his or her own object-oriented solutions, understands the way design patterns work, and is able to use some design patterns in programs.[5] After a while a team of trained and experienced object-oriented developers produce a "greenhouse effect." This means that learning the needed skills is easier for new team members because they can gain knowledge from the experience of their predecessors.

Figure 6 Evolution of programming structures



Goals for the shareable framework department

> *A thought which does not result in an action is nothing much, and an action which does not proceed from a thought is nothing at all.*
>
> —Georges Bernanos

To make the learning curve as short as possible, to keep developers happy so that they continue to work for Consist, and to make it possible for existing customers to grow into these new technologies, the goal for the shareable framework department is not just to build a new product line without looking at what the rest of the organization is doing or, even more important, without looking at what the current customer base is doing.

It is important for the different development labs within Consist to do their work by using the same methods. Therefore, the shareable framework department obtains some developers from other departments and trains them to work in an object-oriented, Java environment. Afterwards these developers go back to their own departments and continue working there in this new way. This approach is a signal to all developers that it does not matter in which development lab someone is working; everyone will grow proficient in the new technologies. The second advantage of this approach is that new modules for the current product lines are developed based on the same technologies that the shareable framework department uses. Therefore, current customers also learn about the new kinds of applications Consist will deliver and can follow an evolving path to the new product offerings. Third, in this way, the learning of new programming paradigms is evolutionary for the organization. This gradual change is of great importance for the stability of the development labs.

The previous discussion may give one the idea that the Consist shareable framework department is just a research department that does not have to deliver any software products to the market. That impression would not be true. Implementations done by isolated research groups that explore new technologies almost always lead to poor choices. Because there is no process for utilizing only a few parts of an entire project when choosing technologies, the mapping of scientific choices on the requirements of the manufacturing process is poor.[6] Therefore,

exploring new technologies is done in cooperation with IBM, other partners, and with the knowledge network for object orientation from Roccade. Roccade is the holding company that owns 50 percent of Consist (the other 50 percent is owned by IBM). Roccade also owns many other Dutch IT companies. These companies share knowledge through the existence of knowledge networks.

In the shareable framework department, the chosen technologies with their subsequent work environment are used in the projects of the department. These projects must deliver either San Francisco-based applications or, at least, Java-based applications that are connected to the existing product line and can be sold to the existing customer base.

To set the right expectations about what can be done at this moment within a shareable framework environment and in a Java environment, it is very important that all departments and employees (not to forget the management team) are informed on a regular basis about the progress made and about the possibilities offered by the technology.

Working with shareable frameworks

The journey of a thousand miles begins with one step.
—Lao Tzu

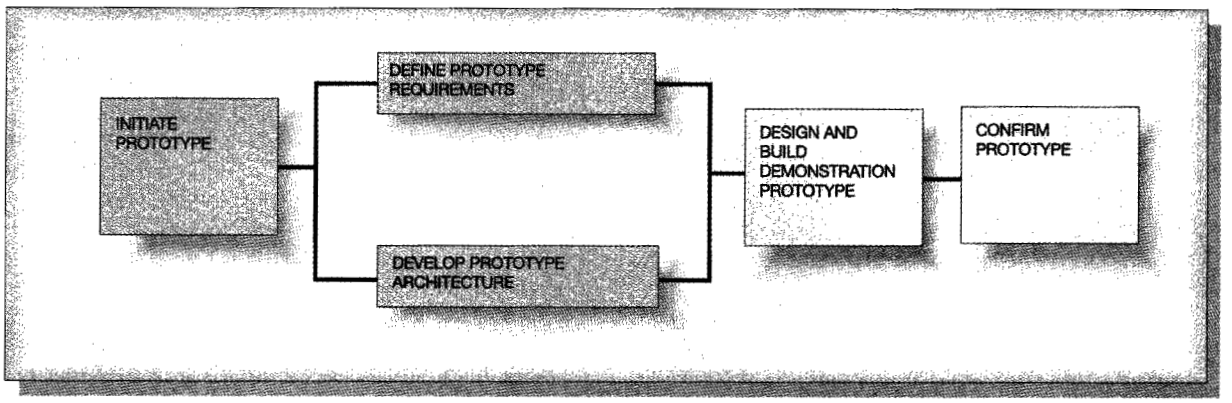
The way in which Consist works in building applications based on the San Francisco frameworks[7] is completely different from the way it is accustomed to building applications. Every new San Francisco developer first has to have much training in object-oriented analysis and design, Java, Rational ROSE**, San Francisco, etc. After that, he or she has to build some Java programs based on a predefined problem to learn how to use Java, some design patterns, and the programming standards Consist uses in this environment. Then the developer starts working in a small team that is building a prototype. Describing the training and the learning stage of a new San Francisco developer is not the intent of this paper. Whole books can be written about that subject. This section of the paper concentrates on how to work in a situation with a trained team of object-oriented developers.

At its initiation, a project is not concerned with the possibilities that San Francisco offers. The first stage was meant to define the trends that could be recognized in the market from a functional and a technical point of view. In order to have these trends well-defined, about 10 project teams were organized so that each had a specific question about the future of Consist customers and business partners. An example of such a question is: "How does a current Consist customer work in the year 2005 in the area of finance?" Each team had three members who could be from any single department, including consultants, sales representatives, or developers. What was important in choosing people was their expertise in a specific area. These teams worked part time to define the trends in their area and delivered a paper on their view of the future. In this way, a picture of the growth expected in general requirements for Consist products of the future was formed. The second stage was called the system model stage. In this stage, the results of the first stage were organized, and a plan was made to determine in which order parts should be developed. After that the actual development process began.

Defining the goals. This process is really iterative. The first project chosen was rather simple. The criteria for the project were: it must be doable; it must have a measurable return on investment; and it must leverage the strengths of object technology. During the first three days of the project, the entire project team was confined to a room where the goal was to have the project defined, the way of working defined, and the project plan created by the whole team; domain experts, developers, and the project manager had to work together for the first time. The goals that were defined were differentiated for the milestones in the project. There are goals for the end of the project (in about a year's time), goals for the end of the year, when a proof-of-concept prototype must be up and running, and goals for two months, when the initial prototype must be ready. The development method that was chosen is that of evolutionary prototyping,[8] shown in Figure 7. With this approach, designing and analyzing the most prominent parts of the program in a prototype are done first, then additions and refinements are made to the prototype until it is finished.

Mapping requirements. After the general requirements have been defined, the requirements mapping is started. For this the "use case technique" is used.[9] This activity is mainly done by the domain experts. The developers have to define the technical requirements. Here the added value of San Francisco becomes clear for the first time. Instead of defining the complete technical architecture in which the application has to run, the San Francisco infrastructure is chosen. This means, for instance, that there already

Figure 7 Evolutionary prototyping



is an interface to map the objects to a relational database, so no time is spent thinking about how this should be done or how to use serialization. There are no worries about which object request broker (ORB) to use and whether this is the right choice. The lowest San Francisco layer has an ORB in it. In the first release this is remote method invocation (RMI). When another ORB is chosen for future scalability reasons, for example, the San Francisco-based applications will keep on running without worrying about this changed ORB, another benefit of San Francisco. An application builder such as Consist must add its own value to topics that it is good at, such as building excellent applications. The more technical activities, such as ORB implementations, can be better left to other companies that are very good at that level.

When a stable integrated development environment (IDE) for Java has to be chosen to build upon the San Francisco frameworks, a little problem arises. All IDEs and all Java development tools are either in beta stage or support an old Java version.

On the user interface side, San Francisco offers great support with a graphical user interface (GUI) framework and a user interface style guide. Currently this style guide supports a Windows**-like user interface standard. In the future an HTML (HyperText Markup Language) standard will also be needed, but for now the GUI style guide and the framework are excellent.

The next step is to identify the application scenarios, provide an abstract for them, and reference the requirements. Each application scenario will be further detailed during the analysis and design activities.

Providing framework support. After the use cases and the requirements for the application scenarios are defined, there first has to be a check of how these requirements map to the San Francisco frameworks. The objective of this activity is to evaluate the coverage and required deviations under the San Francisco framework for all listed application scenarios. The coverage can range from *no* framework support to *full* support. Ideally this mapping should be done by the domain experts. However, since the San Francisco documentation is rather technically oriented and the domain experts are absolutely not technically oriented, there is a problem. Two solutions are available. The first one is to train the domain experts to understand all the technical aspects of the documentation. We absolutely do not want to do that, because the domain experts have to think creatively about what the customer wants on the functional side. If we confront the domain experts with the technical implications of what they design, that may restrict their creativity in the future, because they might feel limited by the technical possibilities. We do not want that to happen. The second solution is to have some developers with the right functional knowledge fulfill a kind of intermediate position between the domain experts and the developers. That solution has been chosen by Consist. Questions that have to be answered during this first mapping process follow:

1. Are San Francisco framework tasks or scenarios available to support the desired application sce-

narios? This may require some navigation through the San Francisco framework documentation.
2. If one or more San Francisco framework tasks or scenarios are available, to what extent do they cover the desired application scenarios? Different cases may occur ranging from no San Francisco framework support to full coverage by the San Francisco frameworks.
3. If the San Francisco framework does not cover or only partially covers the application scenario, what type of deviations are required? The impact is analyzed for:

   –Required user interfaces (changes or new)
   –Required business logic (changes or new)
   –Required information (extra or changed attributes)
   –Other requirements, e.g., legacy interface

This impact analysis will serve as input for workload definition and planning.

Based on the technical and functional requirements, the use case definitions, and the application scenario definitions, the project plan is written.

During the next step the domain experts and the project team together define the first version of the class diagram. Although this version will change considerably in the future, this step is extremely important to make both domain experts and developers think in the same way about what it is that has to be developed. Because the technical implementation details are not important for the domain experts, these are discarded.

An application scenario can be defined as a responsibility or usage ("mini" use case) of the system. The list of application scenarios covers all usages of the system. Application scenarios will be gradually completed and refined. This implies that the mandatory fields and level of detail will change in going from requirements mapping to analysis to design. A decision may be made to keep separate versions for each phase or let the application scenario evolve during the project. Again a mapping of the results of the analyses is done on the possibilities offered by the San Francisco frameworks. The objective of this activity is to extend the San Francisco framework class model so that it can support the services required by the application scenarios. These classes and their responsibilities can be derived from the application scenarios. During the analysis phase it is important to focus on business classes. At this stage no user interface classes or design classes should be added.

An important part of the scenarios in the communication between domain experts and developers is the description of the user interface. We use Microsoft Powerpoint** to quickly define how the dialogue should work and what the screens should roughly look like. The real details of the screen design are made during the programming stage.

After the analysis stage we start with the design. It is hard to draw a clear line between analysis and design. For us the line occurs when a domain expert is modeling; we call it the analysis stage. When a developer is leading, we call it the design stage.

In the first prototype we built, we let the domain experts create the object interaction diagrams. We discovered that this helped them to understand how the system should be working but that they were not able to go into enough detail for the developers. As a result, only the developers are now creating object interaction diagrams per use case. For design scenarios it is important to obtain a level of detail that includes:

- Exception handling
- Reference to used classes
- Reference to scenarios
- Indication of passed arguments
- Pseudocode-like business logic
- Gets/sets
- Interface to legacy systems (if relevant)

The design stage is done almost entirely by the developers. The domain experts are asked regularly about some details that are not completely clear. The design process describes how to translate the analysis model into a design model, taking into account the technical environment, technical architecture (for example, the San Francisco programming model), and performance issues.

We build further on the analysis deliverables using the same techniques. We focus on the requirements needed to use the ROSE-to-Java generator to create Java classes from the design documentation in Rational ROSE. The activities in this stage are:

- Extending the application scenarios with design-specific information
- Designing user interfaces
- Creating a design class model by adding design-

specific classes, relationships, and methods to the analysis model (for example, commands, controllers, collections, and helper classes)
- Restructuring the model to meet the requirements of the technical architecture
- Creating design interaction diagrams to show the dynamic behavior of the application and to validate the design class model

An important part here is the integration with legacy databases: San Francisco frameworks provide for the use of a schema mapper to allow developers to map San Francisco objects to relational databases as their persistent storage mechanism. Also important is the interoperability of legacy code with San Francisco objects: using legacy applications in business objects and using business objects in legacy applications. For Consist, the AS/400 legacy code and databases are extremely important. At the time this paper was written Consist did not have a San Francisco version available that already had this AS/400 legacy mapping. Instead, we used the OS/400* (Operating System/400*) Java Toolkit**. Using the toolkit, we had no problem in going from our San Francisco or Java code to legacy databases and RPG code on the AS/400.

Starting from the design class model (design class diagram, design object interaction diagram, method specifications), we can create and generate the application code (e.g., via the San Francisco Code Generator). Code can be generated from the design model through the use of the Code Generator. In this case the necessary tags have to be added to the classes and methods. Coded classes should go through a unit test (feature test) and code review before they are available for integration testing.

Summary and conclusions

The learning curve that an organization has to follow before it is able to use San Francisco is apt to be long and steep, especially when a company has no experience in working in an object-oriented way. Before choosing San Francisco, a company needs to have a vision of its future and needs to think about the business reasons for making that choice. The main business reason for choosing San Francisco is the expectation that the IT industry and especially the standard application builders will mature in the next few years. As a result, there will be backbones on which all application vendors will build their applications. This means that where application builders now develop almost everything they need for themselves, in the future they will buy components from other vendors and use them in their own applications. San Francisco provides a backbone that is very flexible because, worldwide, thousands of applications will be built on this framework. For customers, the added value is that they can choose the best-of-breed applications for their business out of an enormous suite of applications that will be available on the market. These applications will be connected to one another through the San Francisco base layers.

An organization that chooses San Francisco needs to organize its development labs in such a way that there will be time for developers to pursue the learning curve. Apart from that, the organization should focus on strong human resource management to keep the developers who are not yet working with San Francisco happy. A thorough plan has to be developed indicating what should be done to prepare a development lab to use object technology and San Francisco frameworks.

San Francisco provides a roadmap that describes how to create a San Francisco-based application. Besides the learning curve, the most difficult point is to map the system requirements to the San Francisco frameworks. This cannot be done by domain experts because of the technical character of the San Francisco documentation.

This paper has given an overview of why Consist B.V. in the Netherlands is using San Francisco, what the business reasons for this are, and how the project is organized. It is intended to provide the reader with a high-level understanding of these topics and to show the experience of one ISV in using San Francisco.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Sun Microsystems, Inc., Object Management Group, Rational Software Corporation, or Microsoft Corporation.

Cited references

1. *Software's Industrial Revolution, Enterprise Applications Become the New Backbone of Business*, Deutsche Morgan Grenfell Technology Group, New York (January 14, 1997).
2. M. Healy, "Applicatiepaketten," *Computable* **29**, 37 (July 18, 1997), in Dutch.
3. Taligent, *The Power of Frameworks*, Addison-Wesley Publishing Co., Reading, MA (1995).

4. G. Booch, *Object Solutions: Managing the Object-Oriented Project*, Addison-Wesley Publishing Co., Reading, MA (1996).
5. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Co., Reading, MA (1995).
6. M. Iansiti and J. West, "Technology Integration—Turning Great Research into Great Products," *Harvard Business Review* **79**, No. 3, 69–79 (May–June 1997).
7. Based on E. Callebaut and A. Nilsson, *San Francisco Roadmap*, IBM Corporation (1996).
8. S. McConnell, *Rapid Development Taming Wild Software Schedules*, Microsoft Press, Redmond, WA (1996).
9. I. Jacobson et al., *Object-Oriented Software Engineering: A Use Case Driven Approach*, ACM Press, Addison-Wesley Publishing Co., Reading, MA (1992).

Accepted for publication December 10, 1997.

Rob L. van der Salm *Consist B.V., Nevelgaarde 20, P.O. Box 500, 3430 AM Nieuwegein, Netherlands (electronic mail: robvds@xs4all.nl).* Mr. van der Salm is manager of a department that develops standard software applications based on Java and San Francisco technology and is a member of the Consist management team. He is responsible for the development of new product lines. Previously he worked for SSA Benelux as a technology presales consultant and as project manager for other system companies. Mr. van der Salm studied information science and computer science in the Netherlands before receiving an MBA from Heriot Watt University in Edinburgh, United Kingdom.

Reprint Order No. G321-5673.