



## **e-nvisioning the Future of Application Development**

What do you think of when you hear the word 'Amazon'?

A river? Probably not!

It is difficult these days to read any article on IT without encountering a reference to e-business. As long as anything is related to the Web, it is hot stuff, and there is little doubt that this is going to have a major impact on the way we develop, run, and maintain applications in the future.

The technology change is driving a new mindset in business, a much more holistic view of the business. You are no longer e.g. in the printing business, you are in the business of delivering the work of the artists to their audience. This forces you to rethink your concept of suppliers and customers and forge a new set of business partnerships.

### **IT is Changing Due to the Change in Business**

The salient characteristic of this new environment seems to be an obsession with speed, adaptability, and integration. You need to get there and exploit the possibilities before anybody else, or the opportunity is lost.

This is over and above the impatience the business has always felt with application development. We are not talking about a time horizon of 2 or 3 years but rather of 2 to 3 months- maybe even weeks - and statements like: *'This must be in production before the Christmas trade or we might as well forget it!'* are common. We are not facing a need for a new release every 6 months but more likely the need for a new release every month.

We have had a hard time keeping up until now and we are going to be in deep trouble in the future unless we embrace a radically different way of doing our job.

To make matters worse there are several other problems facing us in this environment.

### **There is a Lack of People with the Required Skills**

There is a scarcity of people with the technical skills needed (e.g. Java etc.) to develop the new applications, and those that do exist are to a large extent quite young people and they might consequently lack training in other disciplines such as requirements elicitation and user interface design.

Trying to install those skills in the young people will meet with a lot of resistance due to the amount of time required to do so and the business-induced need for speed in getting the new systems in the air.

The option of reeducating the veterans in the new skills is fraught with frustrations as those who have been converted to rule-based or object-oriented programming can testify. It takes a lot of time and tears, and there is no guarantee, that they will make it.

Although the new applications are developed in an entirely new context, most of them still need to be integrated into the core legacy systems (accounting, billing, stock management etc.), which are often monolithic and therefore hard to integrate with. Also, the staff familiar with the old applications knows little about the Web side and vice versa.



One company started a huge effort and succeeded in making a website available in time for the Christmas shopping. But they were unable to tie the website to the legacy system for shipping, production planning, and purchasing. This meant that for several months, they were unable to deliver the items they had sold. Needless to say they are no more among us.

### The Requirements are Frequently not well Understood

The requirements are even more nebulous than before. The business people you are working with to define the requirements are no more the users, they are just the people who are responsible for getting to the users.

We are facing a new crowd of users with uncharted habits, who are in no way obligated to use our system and on whom the welfare of our company depends. So we have better get it right, if not the first time, then at least very quickly.

Once you have used the Web yourself, you become aware of how many clumsy, annoying and downright stupid designs there are out here. The User Interface Guru Jakob Nielsen tells us, that if a user does not get to where he wants to go within a site in the third Click, he will leave that site and never come back. So - again - we have better get it right, if not the first time, then at least very quickly.

To make matters worse, the business people are often very unsure about what the requirements should be because the entire environment and its possibilities are new to everybody.

Another issue is, that the communication between the IT staff and the business staff is hampered by the lack of a common vocabulary due to the young IT staffs newness to the business game and the amount of new technology jargon. This makes it even harder to convey the possibilities inherent in the technology to the business staff and the business needs to the IT staff.

But all is not lost.

.

### We Have Some of the Tools we Need

The problem of the moving requirements target and the short time frame would normally be addressed by using Rapid Application Development (RAD) but experience has shown, that while the first efforts using RAD are often a spectacular success, the ROI will decline over time as the problem of getting the hastily thrown together systems integrated grows steadily worse. So RAD alone is not the answer. We need to link that with an architecture that makes integration and sharing easy.

To do that, we can borrow from the concepts of OO and base our architecture on the Model-Controller-View paradigm to create a layered structure of 'Object-like' entities that allows us to be stable and flexible at the same time.

### Standards and Architecture are Major Technology Issues

Many Web-related projects have gotten themselves mired into a lot of inconsistent parts, that are hard to integrate and harder to expand by letting the architecture grow with no clear



vision. This leads to inconsistent user interface and - worse - to steadily slower development as the burden of maintenance and integration sets in.

One dimension to the standards is the need for a consistent look-and-feel of the user interface. This again should be handled by a set of overall principles and some examples, where developers can go and see how to do it. It might also include style sheets to the Web-pages as well as templates and other practical aids, but that is not enough. Given the lack of foreknowledge about how users are behaving, we need to be able to change the look-and-feel constantly. This leads to the need for a function that 'patrols' the user interfaces and allocate resources to fix the problems as our knowledge increases.

This is why an overall architecture should be established as early as possible. It does not have to be complete in any sense, but the principles should be clear and some fully developed examples should be available

This is where the different kinds - or layers - of objects come in. Of course it would be nice to start from scratch and do it right, but do not deceive yourself: you will never get the chance to redo the old legacy applications without a clear and pressing business reason, so we might as well start wrapping them as object-like components step by step. One of the first steps would be to make sure that there are simple 'CRUD' interfaces to create, read, update and delete them, and build them if needed.

Whether this should lead to a *Component Broker* layer in the technical infrastructure, is a question that will probably be answered by the degree of management commitment to the effort. The more serious management is about the effort, and the more sustained you believe it to be, the more likely it is that you will want to create a solution to the integration between the legacy world and the new technologies once and for all and thus create a Component Broker Service available for future development efforts. Conversely, when you are in a experiment- or pilot mode, you would try for simpler ad-hoc solutions.

### The 'Data-Objects' are the Heart of the System

The files / tables - that are normally part of the legacy systems - should be considered model 'objects' and 'encapsulated' by traditional I/O-modul-type interfaces. This is the company's 'gold'. No one should be allowed to do anything to these objects except through the official, published interfaces.

Actually they are more like classes, as only one of each type exists and they have to be supplied with the necessary information to identify the data to work on. So, there is just one 'Customer Object' and the interface for 'Delete' would need to be supplied with e.g. the customer number. These objects handle one file only, by preference, although this is not an ironclad rule but rather an ideal to be constantly strived for.

The 'Model Objects' a primarily service providers (Servers), and the only kind of service requests the will issue is to 'Technical Service Objects' like Authorization, Logging etc.

This is not rocket science, but just good old-fashioned design, and many companies have gone a long way in this direction when they prepared for Y2K and the Euro.



In most cases these object already exists and it is more or less a question of cleaning up the uncontrolled access to the files - in itself not a trivial task - and keeping the using applications supplied with interfaces ('Methods' - to stick with the OO terms).

These objects could be produced in a number of languages including 2. and 3. GL, but in some cases OO-languages combined with an Object Broker would be the optimum solution. In any case Object Persistancy would have to be handled, probably by use of a DBMS.

The creation and maintenance of these objects should be handled by an organization centered around ownership of one or more of the objects and staffed with experienced people from the legacy systems. This part of the organization should be considered as a profit center, which implies, that the using applications should pay for the development, maintenance and use of the interfaces, as establishing a culture of reuse is mostly based on management and financing.

### The 'Control Objects' Contain Most of the Business Logic

The 'Control-' or 'Transaction'-type objects do not necessarily have any data and if they do, it would normally reflect events rather than things (e.g. where a Model Object would represent 'Products', the Control Object might represent 'Claim Verification') and tend to embed business procedures and rules.

These objects issue service requests to Model Objects, Technical Service Objects and each other as needed. They also provide service to each other and to the View Objects, so they have a dual role as both Clients and Servers. By the way, this is the true nature of Client-Serve based systems. Whether the Client and the Server is on the same platform is immaterial to the design, if not to the implementation.

They might be produced in a number of languages and here is the place where rule-based programming would be a likely candidate. They often deal with several Model Object types by using their public interfaces. Many of them will be owned by and internal to the company but some might be produced by customers, business partners or suppliers.

They have their own interfaces, which might be called by internal or external Control or View Objects.

The creation and maintenance of these objects should be handled by organizations with close relations to the business. These groups might well be part of the business organization instead of the IT department. They might be treated as cost- or profit centers according to normal practices in the company, but the service they will require from the Model-groups in order to create new functionality in the Data Objects should be billed to the Control Object Groups.

### The 'View Objects' Correspond to the New Applications

This is the User I/O of the system. It will include Web-based interfaces to the customer (in the CRM-end of the business) and the suppliers (in the SCM end). They can do anything they want to do, as long as they stay within their authorization and only use the public interfaces of the Control Objects and - maybe - Model Objects. They can of course use other View Objects, and could be owned by the company or by anybody else on the Web.



They are first and foremost Clients even though some of them would provide common services to others. They issue service requests primarily to Control Objects but also in some cases directly to Model Objects and Technical Service Objects

The languages here would often be Object Oriented and include HTML and Java as well as a number of more traditional languages.

The creation and maintenance of these objects should be handled by an organization embedded in the Business Organization, for the external functions primarily in Sales, Services and Purchasing. The cost of this should be absorbed by the Business who owns the functions.

Quite a few of these objects would in time be produced by customers, suppliers and business partners, which is why their authorization to access other objects should be tightly controlled. Some of them might even be produced by a new kind of 'parasitic' competitors, who make their business by scanning the market and offer their customers the 'best buy' (e.g. the cheapest loan) thus turning our product into a commodity rather than a premium product. This is a threat that can probably not be contained by IT alone. The Business part of the company should consider this and maybe decide, that if you can't beat them, you should join them.

### The Development Process must be seen in Context.

Regarding a process as an isolated entity that can be changed just by itself is a sure way to fail. We need to look at the different aspects that influence the way we actually develop systems and the way a change to the development process in turn influence them.

The aspects are:

1. *Method*
2. *Technology*
3. *Skills*
4. *Organization*
5. *Measurement*
6. *Culture*

### Method

The development Process must be iterative, incremental and experimenting. The combination of RAD and OO based development procedures seems the right bet for this aspect.

The scoping of the project might be done by defining a number of Use Cases.

They might be assigned to successive iterations / build cycles and in turn tried out on 'users' by a 'low tech' prototype using paper or overhead projector. Once they are ready for production, they should be carefully monitored for actual user behavior and swiftly modified if they do not meet the users' actual requirements.

This is where the 'iterative' comes in.

Managing the workflow and the updates is stated to be the biggest day-to-day challenge in managing the site by 40% of the respondents in a recent survey. So as for *Configuration Management*, you would need to do a major overhaul of the current procedures. The responsibility might need to be distributed and layered.



The changing requirements and the iterative nature of the development process makes changes to the server a very common occurrence. In most cases this will be extensions to current functionality and might classified as:

1. New interfaces (Methods)
2. Added functionality for existing interfaces or Changes to interface layout / protocol

The first case is simple and only requires that the requester is notified when the change is operational (ie. in production) and that the new interface is published in the '*Object Catalog and Usage Guide*'

In the second case the current users have to be notified of the change and have to OK it before it goes into production. An obvious corollary is, that the users of Objects must be registered to avoid surprises, but as they already have to be registered because they have to pay for the use of the service, this should not be a problem.

As there are bound to be numerous objects changing all the time we cannot make do with new releases once every 6 or even 3 months. We need to run regular scheduled Builds e.g. once a week so that a new version of objects always goes into production e.g. Saturday morning. This would be true for the View Objects as well, but in many cases - where the new version of the View Object was dependent on a new version of a Server - it would have to see its service providers go into production one week and then follow them a week later.

The need for an active acceptance of changes from the users makes estimation more difficult, as any change to an existing service runs the risk proliferating to a lot of change requests in the Client Objects and consequently delays outside the scope of the development projects control. This translates into any project doing a thorough Impact Analysis as part of the project definition in order to avoid surprises.

We also need to have a '*IT Management Business Board*' take on the expanded task of prioritizing work, as any prioritization done within a group is likely to be sub optimized. This board would have to meet frequently - not twice a year as is common now - in order to reflect the true and volatile business priorities.

## Skills

The mixture of skills needed is such, that at best you will have only one or two people who have an understanding of how it all hangs together, even on a high level. Trying to make everyone a master of all trades will not lead to success. Instead you will have to divide the competencies between the groups and objects and have a small 'architecture'-function focusing on the overall design, communication between the groups, and on the interface definitions.

## Organization

The Client/Server paradigm must be expanded to embrace the Organization  
Just as the components or objects live in a client / server relationship with each being the customer of some components and the supplier to others, so must the organization begin to see it self. this is logically based on the organization being centered around the objects and besides, it is the way any TQM-effort would like to shape the organization anyway.



This is true for the Achitectual Group as well.

Obviously some new organizations such as Webmaster will need to be created along with a function to manage content of the site. But as long as we focus on roles and responsibilities instead of management turf we should be OK.

## Measurement

In order for the customer-supplier relationship to succeed, you have to make services chargeable as has already been mentioned above. This will immediately focus attention on the activities with the greatest business impact, which is not a bad thing.

We need to be monitoring the users behavior and - if possible - periodically measure their satisfaction and combine that with the dollar inflow of the applications in order constantly to provide the customer with the very thing he needs.

Inside the organization each group should also be able to measure its performance in terms of revenue created from internal customers and their satisfaction.

Finally, the average turn-around time for business initiatives to go into production is a good indicator of our success in mastering the new reality.

So in the end the basic metrics are still speed, dollars, and customer satisfaction

## Culture

The entire culture will be moving towards the object-centered groups, which should be reasonably small, in order for them to become them business minded teams focusing on the needs of their customers, whether they be internal or external. They therefor need to have a lot of empowerment within the larger scope of the business strategy of the company and the architecture of the applications.

Almost half of the people who are already living in that world state lack of initial coordination as a major problem. This should encourage us to plan before we leap and not just get sucked into it by uncoordinated 'try-it-and-see'-initiatives. The need for a management function to control the direction stems from these projects being so tightly coupled with business that neither IT nor business seems able to take control.

This coordination will cut across the 'silos' of today's organization and force Business, Development and Production to team along business subjects and might eventually lead to a lot of the IT department being assimilated by the Business Organization. In some of the more advanced dot.com organizations it is very hard to see where business stops and IT begins.

So IT might go the way of writing. Once writing was an esoteric skill practiced in a special department of the company. That is not the case today.

There are answer to all the problems, we just have to use them

There seem to be real hope that using these guidelines should do a lot to alleviate the problems outlined in the beginning of this paper.

Here is a list of the problems, and the tools envisioned to address them:



## **I. Business Haste**

- Iterative Development
- Timeboxed Development
- Layered Architecture
- Frequent releases

## **II. Lack of skills**

- Joint Application Development
- Layered Application Architecture
- Division of Labor
- Distributed, Teambased Organization

## **III. Uncertain and changing requirements**

- Iterative Development
- Timeboxed Development
- Joint Application Development
- Layered Application Architecture

## **IV. Uncertainty about user interface**

- Timeboxed Development

## **V. Iterative Development**

- Prototyping
- Monitoring of User Behavior
- Frequent releases

## **VI. Communication between the IT staff and the business staff**

- Joint Application Development
- Prototyping
- Distributed, Teambased Organization