1.0 CASE89

1.1 Application Development Trends and Directions

Presentation by Miles Welter, IBM Corporation, Santa Teresa CA, USA.

Let's talk a little bit about software engineering. Software engineering is an attempt to take the same concepts that are used for building a bridge, constructing a building, or designing a computer hardware system, and apply them to the process of developing, maintaining and managing the software that runs on that computer system. These things have been done for a long time and called programming. The advent of software engineering is an attempt to formalize the disciplines that so many people in the programming business just did as an art. They include methodologies and tools. I recall when I started in the business some 33 years ago, my manager called me in and said, It's a shame that you've started into the business at this point in time because we have just come up with a new method that is destined to allow all programs to be produced on time, within budget, and meet all other requirements of the end users. That method was CPM, Critical Path Method. My manager went on further to say that we have a new tool that allows us to fully utilize this new methodology. That tool is called PERT.

Since that beginning a long time ago, many other methodologies and many tools have followed. It's to the point now where a large number of tools is available for the user to select. These tools each promise a significant increase in application development productivity and individually these tools provide that very increase in productivity. However, as more and more of these tools are added, it becomes obvious they were not designed with the other tools in mind. Thus IBM has embraced a new idea to produce a framework for integrating all these tools, and it's this framework and the supporting platform implementing this framework that I want to discuss with you today.

The notion of this framework is to allow a natural extension to the highly successful SAA - Systems Application Architecture - to assist the application developer. Professional Data Processors have been trying to automate the various individual departments for a long time. Now these same principles will help the programmer who has been providing that automation. The basic change to accomplish this task is to take the traditional life cycle curve which starts with an end-user coming in with a problem . That problem is interpreted by the professional data processors and they in turn assemble a team to try to solve that end-user's problem. And there's a very well known set of procedures that go on and on, each individual installation has their own, but they are all similar in that they start out with some kind of requirements gathering, analysis and design at a high level, analysis and design, developing the actual program, testing that program and finally putting that program into production. This cycle is commonly called the waterfall cycle, and I'll talk about that cycle a little later on. That cycle takes a long time; that cycle does not communicate properly between the professional data processor and the end-user. In the end, quite frequently, the product that is delivered, although a product that performs very well according to the specifications that the professional data processors set out to meet, does not satisfy the end-user and that's for a variety of reasons. Frequently the major one of those reasons is that it takes a long time. This long time involves a large number of resources expended and as Software Engineering techniques are implemented, one of the key changes required is to reduce the number of resources expended and that can be done best by shortening the development life cycle.

The secondary thing that must be done is get the application designed properly the first time so that it actually meets the specific requirements of the end user. It is a well-known fact that changes in the application become much more costly further along into the application development life cycle.

Let's examine that life cycle in a little more detail. It starts out with a person in the business recognizing that data processing can assist in reducing the time it takes to do a job or actually in allowing a new job to be done...one that was never done before. That person is typically called an end-user. The end-user meets with the professional data processor and they discuss what they would like to produce. It would be nice if at this point in time a business model could be produced, an enterprise-wide business model, that would provide a pictorial view of the business. A view that businessmen managing the business and the professional data processors attempting to assist by letting the power of the computer help in those difficult areas could visually see how the business was operating. This is an area that is becoming increasingly more popular, but the industry is still not able to handle. Thus, a business model is not created, but instead some sort of specification. Next what happens is a group of professional data processors, highly skilled technicians are gathered together. They might have functions of database design, screen design, report design, specific functional design. These people read the specification that was decided upon by the end-user and the analyst. They in turn produce their own interpretation of that, each aligned along the area of

their speciality. Now instead of one specification, three or four or five different specifications exist, all attempting to get back at the root of what the end-user wanted.

These are given to a team of programmers, and the programmers then produce a program which in turn is tested, integrated with other systems in the organization, and promoted to production. That is the situation today. How can Software Engineering techniques assist in this situation ?

IBM has been working on a notion of a central repository. This central repository would contain control information for all items that were required during the application development life cycle. If such a central repository were available, let's review how the scenario just mentioned would appear. The same end-user would come together with the same professional data processor and they would be discussing now how to solve this end-user's problem. Only this time a set of software engineering tools could be delivered to the end-user by a programmable workstation and the professional data processor along with the end-user can determine exactly what this business model should be. This is going to take a sophisticated type of tool and one that is still in the prototyping stages today, but this is the direction that the entire data processing industry wants to move.

If that business model has sufficient information and semantic vitality, it's possible at this very early stage that some sort of prototyping could take place so that the businessman could visually see what his thoughts transformed into a business model actually meant to the running of the business. It would be possible to iterate through this prototype in such a fashion that the businessman could be quite sure that he had totally communicated everything he wanted to the professional data processor.

Next, as in the previous example, the team of professionals would be assembled. This time however they would be provided with a series of analysis tools. These tools, again delivered on a programmable workstation, would allow the reading from the central repository a machine stored version of the business model. These tools would transform that business model into the specialties that each professional brought to the table. For example the database design, the screen design, specific programming specifications, etc.

The next phase could be to allow a programmer, again operating on a programmable workstation, to read these new specifications stored under the control of the central repository and produce the application, going through testing and promotion into production in a very similar way as stated before.

The major change between these two scenarios is the use of a centralized repository for control of all objects. This in turn facilitates the recording in a machine readable format of the business model and the problem that the end-user is going to solve, and then a series of transformations from that repository controlled information to ultimately produce the program which executes what the end-user had in mind. As more and more tools are made available to professional data processors or even to the end-users, the process will gradually move to the time when that life cycle, that waterfall life cycle that I described in the first part of the paper, could actually be modified. Up to this point in time, the data processing industry has concentrated on providing tools that allow a faster transition of each one of these steps in the application development life cycle. For example, tools have been provided to facilitate moving from the beginning of the programming stage, the actual production of the programming code to the end of that programming stage. These tools are quite familiar to people in data processing. They started out with third generation languages; we're moving now into application generators or fourth generation languages; and on the horizon is the expert systems arena or fifth generation languages.

It's all well and good to traverse these life cycle steps faster, and we the industry have gained significant activity by utilizing these tools. However, to get a quantum leap in application development productivity it becomes incumbent upon the developers of such a system to look at actually eliminating steps in the application development life cycle, reducing the rigor, reducing the amount of documentation required, reducing the number of checkpoints that are necessary to go from an idea to a finished program. I hinted at the beginning of this paper as to what form that reduction would take. It is in the form of a prototyping capability. This prototyping capability must be sufficiently robust to let the end-user, along with the professional data processor, see exactly what will happen based on changes to this enterprise business model.

Once the prototype is agreed upon, in some number of cases, in the early stages of this new type of application business model driven development there will be few times when this will happen, but in some cases the application can go directly into production. As the industry develops more sophisticated tools and the end-users and the professional data processors become more convinced that this is the way to do it, an increasingly larger number of applications should be able to go from prototype into production.

Let's talk a little bit now about the platform and vision that must be necessary for a series of tools to work together in such harmony as I've depicted in the early part of this paper. Three things must be present. There must be a repository, a very full function repository that will facilitate the control and sharing of information in all phases of the application development life cycle. There must be a complete set of user tools. These tools over time need to cover the entire application development life cycle, and they need to derive the information that they require from the repository. And on top of all this, there must be an efficient, easy to use, and consistent interface into this environment. Those three things I want to speak about in a little more detail.

First the interface that the user would have to this. Let's call this the user services interface. User services is the glimpse that the user has of this environment for increased application development productivity. This glimpse he has should be exactly the same no matter what tool he's using, what function he wants to invoke and no matter what phase of the application development life cycle is currently in operation. Thus, the user services can be broken also into two pieces. The dialog interface (that is the specific interface that the user will see) and the actual tool invocation, because all tools should be invoked in exactly the same way. IBM has put into the market two products that can assist in this area. The first one of these is a product that runs on the main frame and is a system controlled way of driving a methodology and invoking tools. That tool is called ADPS (Application Development Process System). It's a very rigorous product. It requires a fair amount of effort and time to fully utilize. But once installed, will provide a significant application development boost.

The second product is the Professional Work Manager product. This is a product that operates on a programmable workstation and allows the user to determine what sequence is required to execute the application development life cycle. The Professional Work Manager, once this sequence has been determined, executes this sequence over and over for the programmer.

Next, the repository, or the base on which all of this is built. The repository to assist in application development must be able to handle the entity attribute relationship way of storing and controlling those things that are necessary for the application development life cycle. This ER, as it's frequently called, way of storing data is the one that is coming forward as the standards controlling application development gradually matures.

Since object management is also important, a repository should have the ability to take a step up and handle a larger aggregate of entities frequently called objects.

To insulate the builders of tools from changes and also to provide the ultimate in flexibility, multiple views of this information is necessary. The conceptual view, or the view of the entire business enterprise, is the foundation of the repository based application development system. Each tool that uses the repository or that references this conceptual view looks at this view in a slightly different way. Therefore each tool needs to have its own logical view of the overall business wide conceptual view. These views ultimately result in physical data being stored...physical entities, physical relationships and attributes. This data is stored in yet a third view...the storage view. Given these three views and given that the repository is able to handle entity attribute relationship and object management data, the start of a system of sharing of information between all tools is possible. This brings us to looking at the third piece of this environment....that of the tools themselves.

A wide range of tools is necessary to completely fulfill this dream expressed in the earlier part of the paper to allow a very easy migration from a change in the business model to a brand new program. It will probably be a good long time before a complete set of tools is available. In the meantime, existing tools from a variety of vendors, both IBM and independent software vendors can be modified to utilize a repository based development environment.

This modification comes in two directions. First, conformance to a standard way of accessing the tools. Accessing means the way that the user looks at the tools and the way the tools are invoked. Looking to SAA for guidance, the Consistent User Access interface or CUA interface is an excellent interface to take as a pattern. It spells out exactly how items should be represented on the screen and could be extended to indicate how tools should actually be invoked. At the other end of the spectrum is the Common Communications Support. This facilitates the communication between workstations and the SAA host platforms. These host platforms are MVS, VM, OS/400 and OS/2. The view into the entire SAA environment is through a series of Common Programming Interfaces. Those Common Programming Interfaces (CPI's) ensure that a program written in any system as long as it abides by the SAA common programming interfaces or CPI's will run in any other environment. There are 11 currently declared interfaces, and those interfaces range from Fortran to SQL to CSP to RPG. A natural extension of those interfaces would be to provide an additional common programming interface or CPI to support repository services.

Given these two kinds of interfaces, CUA and a repository services interface, the tools currently in the marketplace can gradually migrate into this environment by adopting all of the requirements of these interfaces. It is anticipated that this adoption will take some time and that a starter set of tools needs to be provided by IBM. These tools will most likely consist of IBM products and some independent software products and will gradually be expanded to cover the entire range of the application development life cycle.

Once this has happened, the picture you can visualize for application development would be a separate application development system with a cooperative model, all interfaces being done on a programmable workstation, the PS/2 with OS/2 Extended Edition, and interfacing to any of the SAA hosts. This environment would in turn produce target applications that could run on any of the SAA environments.

In summary, I am looking to a new way of developing programs. A way that is model driven. A way that will involve the end-user early in the life cycle and through prototyping ensuring he is getting what he wants. A repository based development environment will facilitate the sharing of information between tools, and as new advances come, these new tools can be inserted into the environment in the same way the existing tools are.

If all of these things come to pass, then the first steps toward changing the waterfall cycle and eliminating actual steps can take place. The idea of a business enterprise model can be established. The SAA application environment can be extended so that a user is sure that if he sticks to the interfaces that are published, he will be able to take advantage of all changes in technology that are coming in the foreseeable future. It may possibly get to the stage where programmers in the traditional sense as seen today, that is handcrafting a specific program to meet a specific set of requirements, will become a thing of the past. And, the programming talent that exists today will be marshalled to produce better and better tools, all integrated on a common repository, and all allowing that huge backlog of systems to be implemented in a very quick and efficient way.