The Role of Knowledge Based Systems to Enhance User Participation in the System Development Process

Gian M Medri, PKBanken, Stockholm

Summary: Computers are a fact of life today, even for the public in general. User participation in the system development process is made easier by the present development of technology. Acceptance of the users' claim to control their computer systems is mandatory not only from a moral, but also from an economic point of view.

It's obviously helpful to structure the problems related to the development of a new system in a reasonable way, for example by dividing them into inherent problems, non-inherent problems and symptomatic problems. Examples are provided.

There are, at present, three main technologies of application development. Of these, expert systems represent the highest level.

As a conclusion an outline of the ideal knowledge based system development tool is presented. This tool does so far not exist in the form of a CASE package, but it would be quite feasible to make one.

The main problem today is certainly not the lack of visions and results in the field of technolgical development. It is rather that most data centers work for and with third generation software. A CASE tool of the kind outlined here would provide the possibility of enhancing user participation in the system development process. One important reason for this is that it might be of great help in cleaning out obsolete technologies and working methods from the data centers we have at present.

Background

Computerization has already brought about profound changes in everyday life. No doubt the new technologies of Artificial Intelligence will be reflected in new social structures. This is, however, a slow process, due to social inertia, and we know very little about the outcome.

What we know, instead, is that, partly as a consequence of the development of fourth generation languages, new tools for prototyping, the widespread use of personal computers etc, there is a growing claim from people working with computerized systems to participate actively in the design and solution of their systems.

The computer is today a common tool, used even in private homes. Only a decade ago we counted the computers in the world in thousands, now we count in millions.

This means that almost everybody has some contact with computers. The computer does no longer appear as something out of a science fiction novel. An increasing number of people do not accept to be told that it's the computer's fault, when errors appear in their bills or bank statements. They know that the error generally is caused by the people who made the program. This is, of course, a very positive development.

Earlier, computer systems appeared mysterious to the man in the street. Users needed special intermediaries, in the form of system analysts and programmers, who interpreted and coded the knowledge necessary to solve the user's problem.

Today, with the introduction of spreadsheets, 4GL, WYSIWYG (what you see is what you get), WIMPS (windows, icons, mouse, pulldown menus) and knowledge based systems, the handling of computer systems has become much easier.

The acceptance of the users' claim to participate in the development process and control their computer systems is today mandatory, not only from a moral but also from an economic point of view. Nobody can work efficiently and be satisfied with obsolete technologies. Experience shows, for example, that users of a word processing system find it difficult to go back to the typewriter, almost as if they were offered a clay tablet or a piece of calfskin instead of writing paper.

Possibilities and problems

Rapid changes in the market place determine the need of both active user participation in the application development, and a rapid development process. This means that the user must understand the whole process.

Knowledge based systems seem today to be the best way of meeting the requirements of easy-to-understand system design and of quick solutions, provided that proper tools and methods are connected to the expert system for system development.

Some of the CASE (computer aided system engineering) packages in use today are excellent tools in the hands of system analysts with many years of experience. But in the hands of system analysts who lack sufficient experience they can create more problems than they solve.

I have seen several cases where such tools, improperly used, produced hundreds, in some cases thousands, of nice laser printed logical diagrams describing functions, processes and information relations. After that,

nobody understood what to do or how to continue to get a workable system off the ground. This was due to the fact that there was too much information and too many people to get a proper structure. In some cases the project was stopped after many person-years of work.

In the world of manufacturing, we are used to handling concrete things like boxes, wheels, engines etc. When we try to transpose the construction process from classical manufacturing to software we often make mistakes.

There are, for example, still many Information System managers who think that when a project is late, it is sufficient to put more programmers or analysts to work, just as a production manager would do. When he has to double the production of nails, he doubles the productive resources. If one person produces 1000 nails per hour, then two persons should produce 2000 nails per hour.

This linear relation between workforce and production is generally correct in the case of hardware. It is absolutely wrong when it comes to software. In many cases we can observe the strange phenomenon that the input of more resources into a project will slow down if not kill the project itself.

Structuring the problem

Before deciding which is the best way to develop a new system we must structure the problems related to the process. The problems can be divided into:

- **inherent problems**
- **non-inherent problems**
- **symptomatic problems**

The inherent problems are those created by the application itself. For example, a bank with 500 branches is to develop a system for updating balances. The inherent problems in this case are the matching of balances and the amount withdrawn, the interest calculation and so on.

By non-inherent problems I mean problems that are created by technology, policies or environment. An example is that 500 branches generate 100 transactions per second and that the actual balance is in a subrecord of a hierarchical data base. The amount of transactions generate the problem of response-times and special solutions must be found to make them acceptable. Complex solutions are also needed in order to navigate through the data base to find the right subrecord.

The symptomatic problems could be that the development is slow, the project costs exceed the estimates, and that a specialist for the response-time optimization is not available.

In this example we can eliminate the non-inherent problems by distributing the data base to the branches, diminishing the transactions rate to one per five seconds, and substituting a relational data base for the hierarchical one. We can perhaps use a fourth generation language, so we won't have to bother too much with technicalities, and concentrate our resources on the application.

By doing this we also eliminate the symptomatic problems. This situation, especially in more complex form, is fairly normal in the system development environment.

Knowledge based development systems can deal with this kind of problems, if a very experienced system analyst is available.

Who is the expert?

To avoid misunderstandings, I must explain that i call "**experts**", those who are experts in the field covered by the application, or those who represent the end-users. "**System analysts**" are obviously those who are specialists in system development. Furthermore, I call "**system tools**" those systems or packages that help system analysts and experts to develop systems for the end-user and "**application**" the object of the system development.

Very often the non-inherent problems are of technical nature. They complicate not only the system but also the user's understanding of the solution. The power of a CASE package based on an expert system is among other the possibility to help the system analyst to simplify each stage of the system development process, while at the same time the expert can follow and determine the rules that govern the application. The almost natural language used for the application development is similar to the language used in the tool that governs the methodology for system development

The Importance of Flexibility

The system development process consists of many phases that are not cleary defined and change in time and space depending on the technology used, the kind of industry and the local environment.

This means that system tools must be very flexible and adaptable. A knowledge based system tool can easily be adapted to any working place, because the knowledge base is understandable and furthermore separated from the inference engine.

Normally, there is a common base of knowledge for system development that doesn't differ from place to place or time to time. For example, the rules that govern normalization of information are the same independent of the country, the industry or the technology used in a particular development process. Perhaps in the future someone will find some new normalization forms, but the fundamentals are there and will not change. Furthermore, many rules of the thumb in such a system tool will not be subject to change.

Rules of thumb

Just to give an idea of such a rule of thumb I will mention what happened when a new system was taken into production and after running a few weeks showed that many numerical fields did not balance. When we looked at the system specifications we didn't find anything like balancing the sum of numerical input fields with the corresponding changed fields. When we asked the system analyst why there were no balancing checks in the different modules and programs, he answered: "We have been taught that the computer never makes mistakes. It is a waste of time and other resources to include these unnecessary controls."

It is true that undetected hardware errors are very rare, but it is not the same for software errors. It is far from unusual that a program has fifty or more two-ways branches. This means that the number of combinations is a figure with fifteen digits. Consequently, it is impossible to run a full test of a program of this kind.

The system tool should contain the rules of thumb that handle this kind of problem and advise the system analyst and the expert at the proper phase of system development.

The examples described above are of a general kind and can be standardized in a package, but there are other aspects that cannot be generalized because they depend on the country, the kind of industry and ultimately the individual company.

The knowledge system tool should be made in such a way that it is easily completed by the individual company, so as to include their own particular rules.

The most obvious adaptation is the language of the country. The next is the particular business language used in the industry. For example, the terminology of banking differs from that of engineering or insurance.

The language and terminology aspects are very important for the communication and the understanding between the system analyst and the expert. If the system development tool is easy to understand, this characteristic will usually hold even for the application developed by means of the tool.

Another domain where the adaptation to the policy of the individual company is very important is the security and auditing features. The knowledge system tool should emphasize this sadly neglected problem. Part of the rules can be generalized, part of them must be adapted to the individual company.

Three technologies

At present, there are three main technologies of application development:

1. The old one, with static life cycle, with COBOL or some other programming language at the same level, and hierarchical or network data bases.

2. The fourth generation packages, with an iterative life cycle, a very high language level, form, graphic, and report facilities and a relational data base.

3. Expert systems with an iterative life cycle, knowledge bases (rules, demons, facts and frames etc.) and a relational data base.

The system tool should support all the three types. Since the highest form of application development is knowledge based, it seems obvious that the best support will be a knowledge based system. This combination - system tool and application development made with knowledge based technologies - will greatly facilitate the communication and understanding between the system analyst and the expert, and this will contribute to a better system quality.

The second alternative is also acceptable, because the expert can follow the iterative development process and see what happens, so he can check that the system will be what he wants.

The first alternative is not so good, since there is no way to make the system easy to understand for the expert. It is impossible both for the expert and the system analyst to foresee all the logical consequences of the system design.

A knowledge based tool will help the system analyst and expert to formulate the requirements in the form of rules and facts that can be tested preliminarily in an expert system and then translated into COBOL or other similar languages.

In this case, the knowledge should be limited to rules, demons and facts. Frames, probabilistic reasoning and other advanced knowledge techniques should be avoided, because it would be difficult to implement them in COBOL. Relational data bases

should be preferred to hierarchical or network data bases. The more the expert understands the application development, the greater will be the final usefulness of the production system.

Conclusion

Since applications are systems that support human beings at work, whether they be bank officers, medical staff, or spare parts service clerks, the usability of the system must be the highest possible.

To reach this goal, those who are really experts in the field to be covered by the application should participate in and control the application development. The ideal knowledge based system tool would give the best support to application development, for the following reasons:

- **High flexibility with the possibility to implement local specialized rules and facts, language and terminology.**
- **High adaptability to different kinds of development that depend on the environment.**
- **Rules of the thumb extracted from a very experienced system analyst.**
- **Advice to the expert and the system analyst on very complex matters.**
- **Calculations of the risks of omitted or limited security.**
- **Fuzzy and probabilistic methods that can be used when the boundaries are unclear, for example the question whether a system is complex or not.**
- **Easily understandable development process, due to the use of rules and local terminology. The decision makers and the user representatives can follow, understand and control all the phases of the development process.**
- **Possibility of explanations for every advice and decison.**

As far as I know, there is at present no CASE package that really covers all the points mentioned above. It is, however, perfectly feasible to make one, and better tools will certainly be forthcoming within the near future.

There are many visions about future research and development in the computer world. The sixth generation already exists in the minds of some researchers.

Unfortunately, there are not many visions about the use of existing technology. Most data centers of some size are doing more than 90 percent of their work with and for third generation software.

Would a knowledge based system tool of the kind outlined above help to bring the computer community up to date?

My answer is: not totally, and not at once. But it would certainly be a step - maybe a giant step - in the right direction.