INTEGRATED MODELLING AND SUPPORT ENVIRONMENTS FOR INFORMATION SYSTEMS

A SOLVBERG

Rapporteur: M Elphick

Paper presented at the 23rd Newcastle-upon-Tyne International Seminar on the Teaching of Computing Science at University Level

INTEGRATED MODELLING AND SUPPORT ENVIRONMENTS FOR INFORMATION SYSTEMS

Arne Sølvberg
Information Systems Group
Dept. EECS, The Norwegian Inst. Technology
University of Trondheim
Norway

The invitation to this seminar states that "the role of the speakers will not be to give conventional research seminars, but rather to give their views on the present state of the art and of probable future developments in their particular area of interest and expertise, and where appropriate to discuss what part material from this area should play in computing science curricula, and how it should be taught". I shall try to keep to this view of the role.

Information Systems Engineering is the discipline of developing and maintaining computerized Information Systems. In the past these systems have mostly been developed in a "tailor-made" fashion. Companies have developed their own in-house systems from scratch. This practice has led to increasing maintenance burdens on their DP-departments, and to embarrassingly high information systems expenses.

Some of the changes that are taking place in system development practices may be characterized as follows:

- the software profession is slowly maturing from garage ventures into an industry
- an appropriate management culture and methodology will evolve for DP-departments, as well as for software houses
- software development is becoming increasingly capital intensive
- software engineering environments will become a costly must
- a number of prescriptive software engineering standards will emerge
- software management and maintenance will increasingly become major problem areas, and will necessitate the use of rigorous CAD-methods in software design
- the entry ticket price to the professional software market will increase strongly

There is currently a clear trend towards increased sharing of development- and maintenance costs of Information systems software. Systems are no more constructed from scratch. They are increasingly assembled from available software components, which are interfaced and integrated into systems. Vendor-supplied application platforms will appear in the marketplace for more and more application areas.

An information system may be viewed as consisting of three layers:

. the organisation layer
. the application software layer
. the computer system layer

Information system design comprises organisational design, application software design, and computer system configuration, plus appropriate interface design between the layers.

An information system designer will have to participate in designing all three layers. Even if appropriate design skills relevant to all three levels are needed, specialisation to the different levels should be expected. Some of the relevant skill-areas are

. business analysis
. socio-technical design
. human-computer interface design
. functional analysis
. software design
. database design
. performance evaluation
. communication system design
. computer system configuration
. system administration
. project management skills, and so on.

Some of the most important developments that will influence Information Systems development techniques in the years to come are found in the realms of:

. application platforms
. system development environments
. integrated specification models

An application platform contains basic software functions for a particular application domain, plus possibilities for the platform user to add on company-specific software. An application platform is a vendor supplied "common system". It is intended to comprise 40-80% of the software that is required in an information system within some particular application area. The remaining 20-60% of the application software define additional behavioral properties of those information systems that are based on the platform. The add-ons therefore define the competitive edge of the respective organisations.The basic idea of a common application platform is indeed an old one. The new development is that finally the idea seems to be able to fly.

A system development environment is a particular type of platform, which provides an infrastructure for the integration of available system development tools, beyond the conventional level of compilers and database systems. A most important component of system development platforms is the system encyclopedia (data dictionary, repository, specification database), which makes it possible to integrate the various application systems of a company, so that it may become possible to obtain control over the evolution of a company's software.

An integrated specification model makes it possible to formally express a system's properties on every level of abstraction, from the business policy level to the operational data processing level, such that the specifications may be formally massaged and analysed. "Automatic programming" based on functional application systems specifications may be within reach.

Developments like those indicated above will contribute to promoting programming from being an art to becoming an engineering discipline. The preferred talents and skills of the programmer may be different in the future from what they have been until now.

The three realms will be discussed in some more detail below.

1. APPLICATION PLATFORMS

Vendor supplied software are usually found in the form of software libraries e.g. mathematical subroutines, or standard software systems e.g. spreadsheets, wordprocessors, simulators. The software is usually void of domain specific semantical contents. It may therefore be used in a variety of different users domains, e.g. a wordprocessor may be used by both medical doctors and solicitors.

An application platform is a vendor supplied collection of software for some application domain, which may be augmented by the customer to suit his organisational needs. An application platform provides a "core" of software components for standard business functions, plus pre-defined interfaces for custom-tailored add-ons. Application platforms are similar to so-called "common systems". The major difference is that a platform is supplied by computer vendors and software houses, while "common systems" are usually developed by a single large company to be used by its many subsidiaries, or by a group of similar companies in order to share software development and maintenance costs. Examples of platforms/common systems are airline booking systems, banking systems.

The rationale for developing an application platform is economy of scale. Information systems are becoming more and more complex and expensive to develop and maintain. It makes sense to share costs through standardisation of business functions and software. We may therefore expect to see more platforms in new domains in the years to come.

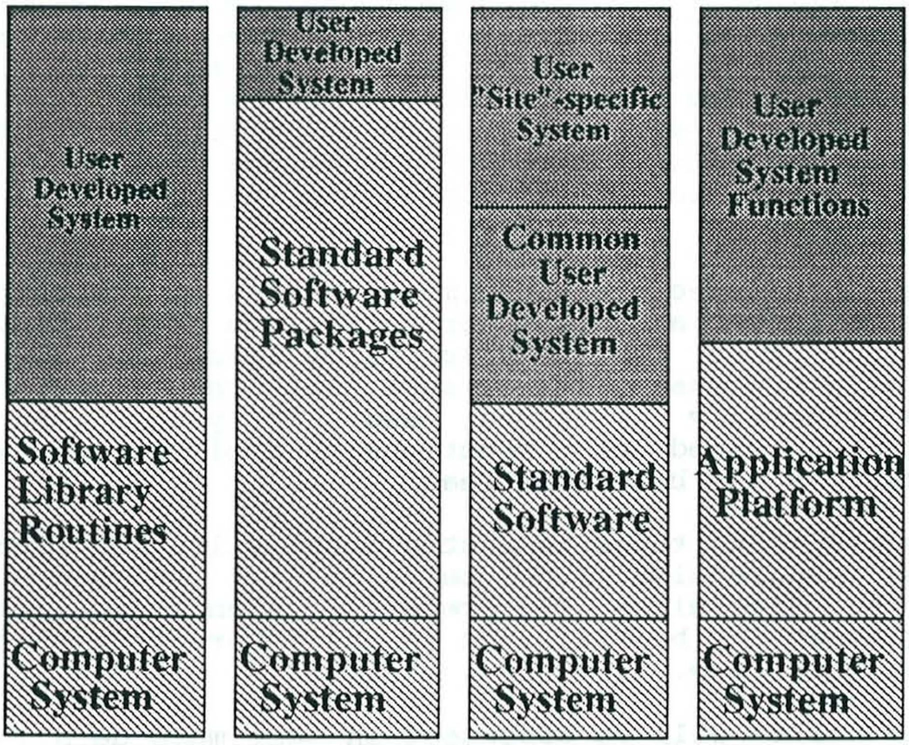


Figure 1 Four types of relationships between software vendor and buyer

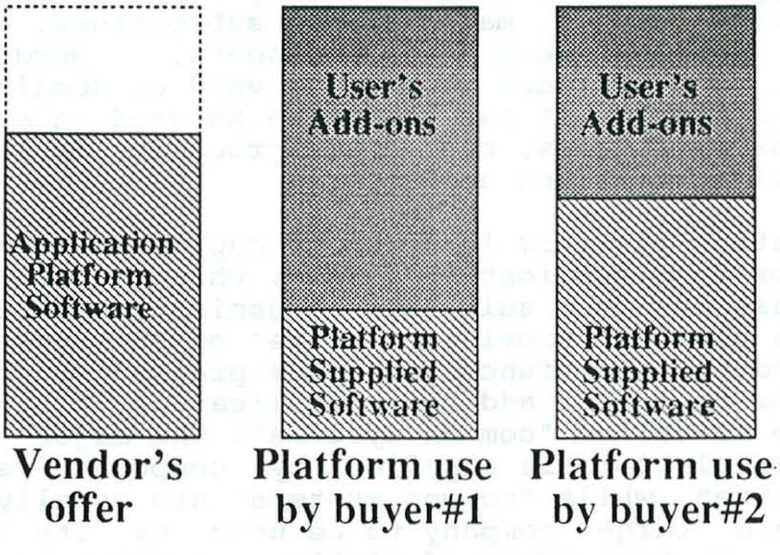


Figure 2 Different buyers may use various parts of platform supplied application functions

An application platform must have an architecture that accomodates the need for modifying a standard solution of a general problem, to fit the local problem at hand. A platform contains a "core" and local add-ons. The core is supposed not to be modified, and is therefore to be treated as a standard software system.

The key question when designing the core is, which functions to standardise in a typical user environment. Is the core 80%, 60%, 40% or 20% of a typical installation? It can certainly not be 100%. In that case every bank would be the same if they used the same application platform. Business practises are embedded in the application software. The competitive edge between businesses is found in the local add-ons!

Application platforms may either be database centered or program centered. In a database centered view, the platform is viewed as a database on which programs are hung. In a program centered view a platform is a collection of programs. In the program centered view little attention is paid to the consequences of changing the database. This may easily lead to incompatible databases at the different installations. If a company has several installations of the same platform in different subsidiaries, system integration goals may become unattainable. One may therefore expect that the successful platform architectures will be database centered.

Even if application platforms are domain oriented, they will have to contain a number of general data processing functions, e.g. word processors. This may lead to problems with the integration of a platform with other software packages that are already used by the platform buyer.

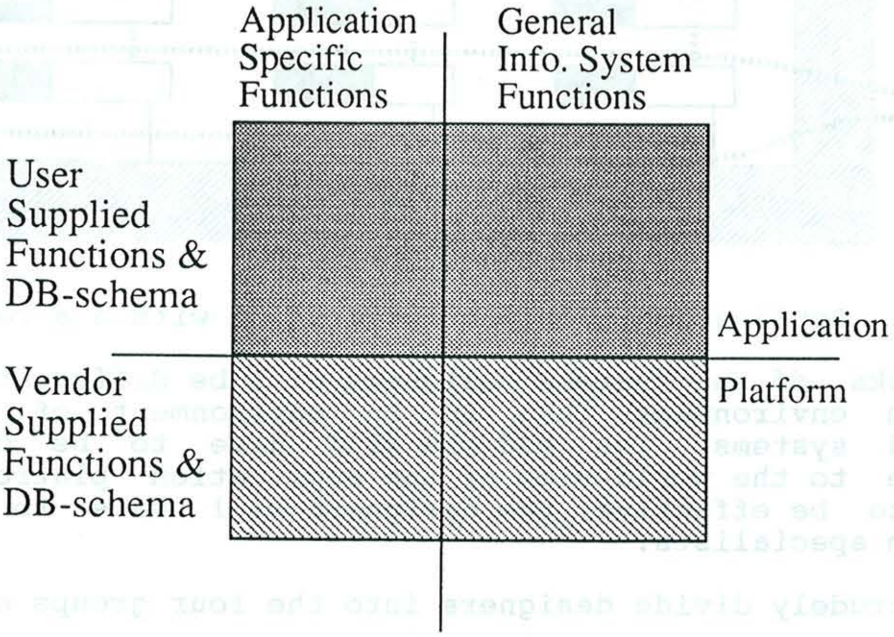

Application Specific Functions
General Info. System Functions
User Supplied Functions & DB-schema
Vendor Supplied Functions & DB-schema
Application
Platform

Figure 3 Application platforms may contain general office functions as well as domain specific application functions

The platform buyers - the companies - will face additional problems. Integration of information systems applications is also today a formidable challenge. The databases of the many information systems that are in operation in a large company reflect various views of the company and its environment. Systems integration requires that the databases are integrated. This requires that a common world view is developed within the company.

Even if a company is completely free to determine their database schemas, independently of commercially acquired application software, it is a formidable job to develop a common conceptual datamodel for the whole company. One of the reasons for this difficulty is that those who know enough about the company to be able to form a consistent worldview of the company and its environment, usually have been in management positions for such a long time that they are no longer candidates for doing systems development work any more. Software development is the battle ground of the young and vigorous, who know enough about computers, but too little about the overall operations of the company to be able to do a proper integration job. This problem is not going to become one bit easier to solve when application platforms carry their own vendor supplied world views into the companies!

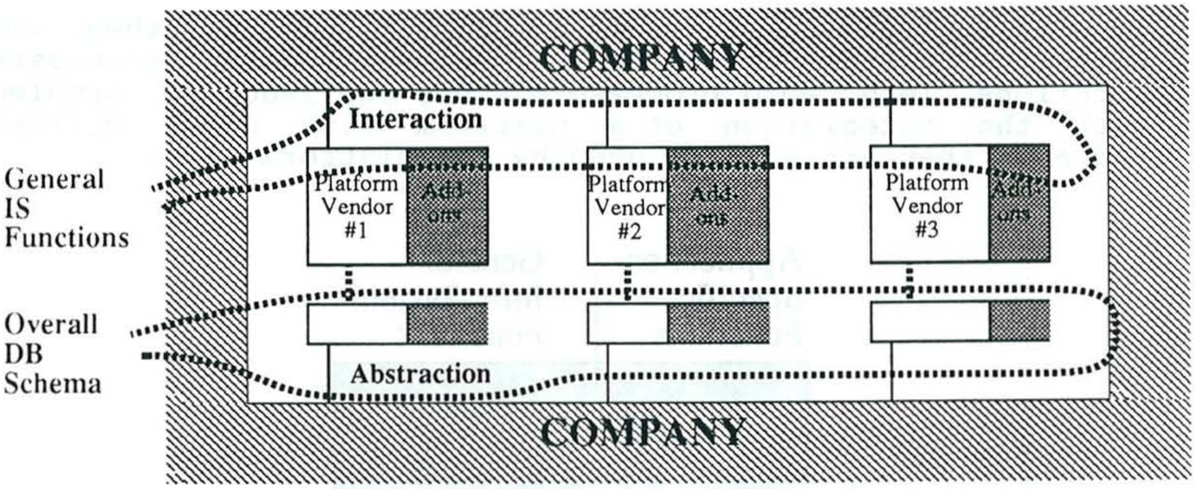


Figure 4 Application platform integration within a company

The tasks of the systems designers will be different in a platform environment than in an environment of custom-tailored systems. The add-ons will have to be designed relative to the world-view of the application platform. In order to be effective, the designers will have to become platform specialists.

We may crudely divide designers into the four groups of

- computer system designers
- application platform designers
- application system designers, and
- organisation designers

What are the appropriate skills for the four groups of designers? And who shall educate them?

Computer system designers are straightforward to place. They are the responsibility of the Computer Science Departments (and the Electronics Departments, and the Telecommunication

Departments?). The other three categories are more difficult to place. Should designers of Office Automation platforms be educated in Business Schools or in Technical Universities, or both? And what about Civil Engineering platforms? Do everybody who participate in the design and implementation of a software platform for road design need to know very much about road engineering? And on the contrary, does a road designer need to know very much about his software platform, except of its functionality?

Is it possible to design an organisation in the future without having profound knowledge of design options for its computerised information system? That is, office system platform knowledge may be crucial as a basis for determining organisational structures.

The organisation of the educational efforts shall, in the long run, have to be determined by the skill sets that are needed for the various professions, and how they can be most rationally provided in teaching.

2. DEVELOPMENT ENVIRONMENTS: INFORMATION SYSTEMS SUPPORT FOR INFORMATION SYSTEMS DEVELOPMENT

Information systems used to be built from scratch, because there were few, if any, components available to base the new systems upon. This is not the case any more. Information systems are increasingly being built through the re-use, modification, integration and interfacing of commercially available software components. Programming, in the original meaning of writing commands for a computer to follow, is not as dominant an activity as it used to be only a few years ago.

More of the total effort of a development project has been shifted from the actual writing of the software, to finding out which software to write. This change has come about because too many projects failed, in order to secure that the software is useful for some worthwhile purpose when it is finally written.

Furthermore, because the information systems development process is team oriented, rather than individual oriented, much effort has to be used to ensure effective communication within teams and between teams. Much effort has also to be used in order to ensure that team members know enough of what is going on in their projects, so that they do not make wrong design decisions out of ignorance.

Important trends in this change of the information systems development process, from relying on programming skills alone, to emerge as an engineering design discipline, are:

* Increased need for practical ways of managing the many persons and tasks that make up the cooperative process of designing and building an information system.

* Increased need to move from ad hoc development strategies to a strategy based on engineering principles, e.g. the development of standards for programming and for specifications.

* Increased availability of a larger variety of computerized tools for the support of the various systems development tasks, in addition to the well known tools for the support of the programming task, e.g. tools for diagramming, code generation, testing, verification, and so on.

The many persons/many tasks problems are enhanced when the information systems engineering department is given a geographically decentralised organisational structure. The tendency to decentralize operations has increased in most enterprizes over the last couple of decades. The information system engineering departments are no exceptions to this trend. Decentralised systems development environments must therefore be supported. Particular emphasis must be given to improved communication, in order to enhance the individual information system engineer's level of understanding and knowledge of the system to be built.

In spite of the recognition of the need for standardization, there is a plethora of available development methods, specification techniques, and programming languages. There is, so far, no indication of that the industry, as a whole, will settle for a broad agreement on standards for information systems development in the forseeable future. Therefore, support systems must be able to incorporate various techniques and tools as they become available and are offered in the marketplace.

During the 1980's a large number of so-called CASE-tools have been developed and marketed. CASE is an acronym for Computer Assisted Software Engineering. Most of the tools are based on system development techniques that originally was intended for manual use by humans, e.g. data flow diagramming. Few CASE-tools have so far taken advantage of the potential of doing more sophisticated analysis of the specifications, provided that the specifications can be more formally expressed. Many of the CASE-tools provide graphical support for drawing diagrams only. Their usefulness is therefore quite limited.

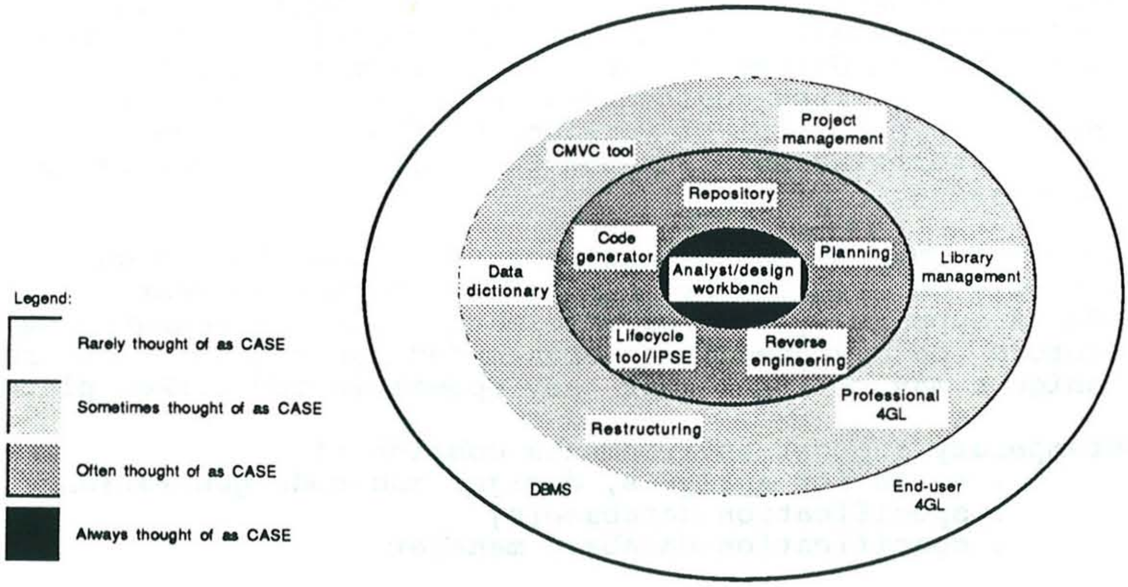


Figure 5 What is a CASE tool

Given these trends, we may propose some requirements which a support system for information systems development should be expected to satisfy:

* The scope must be so wide as to support system development and maintenance through all phases of a system's life cycle. Various system development methods, project sizes, and system types should be supported equally well.

* The project management aspects of systems development must be supported in a flexible manner, so that the support system can be adapted to whatever management strategy that a company may choose to implement.

* Relevant project information must be supported in such a way that it can be made available to all project participants, as needed. Project information is to be understood as software, systems documentation, company standards, company adresses, tools indices, and the like.

* Easy communication among project participants must be facilitated. This comprises end users communicating with the information system engineers, designers communicating their perception of end user requirements to the implementers, the communication among designers and among implementers, and finally, the transfer of knowledge of the whole program system, on every abstraction level, to the staff responsible for the future maintenance of the information system.

* The re-usability of software components and the associated documentation must be supported. The development of modularised software systems must be supported and stimulated. This means in particular that solutions to the problems of the interfacing of software components must be provided for.

* Tools must be provided for effective software configuration management, and for version control of associated specification documents. This is of particular importance in a decentralised engineering environment, where the need to keep control of versions of design components, their interrelationships, and the status of each version is pressing.

Several of the preceding requirements go well beyond what can reasonably be expected by the current state-of-the-art (as of 1990). A support system for information systems design should therefore be designed in an open-ended manner, so that new techniques may be applied as they appear in the market place.

Contemporary support environments consist of

- tools for analysis, design, and code generation
- specification database(s)
- specification database manager

The various environments differ in how integrated and comprehensive they are with respect to the system development process. A most comprehensive specification database may consist of four (overlapping) clusters of specifications:

- a "world model" of the target system and its environment
- a model of the software and data of the application system
- a model of the various configurations of the application
- a model of the development organisation

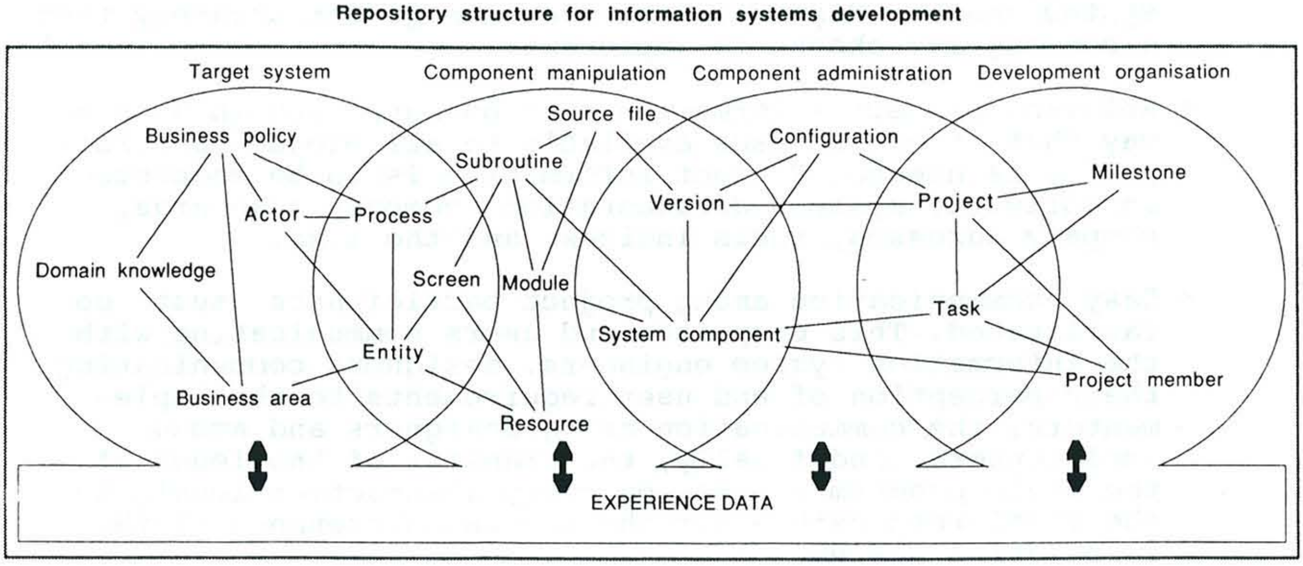


Figure 6 Repository structure Information systems development

There are several ways of organising the tools into toolsystems. Individual tools for analysis, design, code generation carry their own specification databases which are based on different views of information systems. These tools are mostly incompatible. The output of an analysis tool can usually not be used as input for a design tool, without being transformed and augmented by a human designer.

Analyst/designer workbenches organise tools for target domain modelling, requirements capture, software and database design within a common user interface. Most workbenches support several overlapping, incompatible tools. The specifications captured by the tools are stored in the workbench data dictionary. The granularity of the data dictionary may differ from document level to a more integrated approach, in the various workbenches.

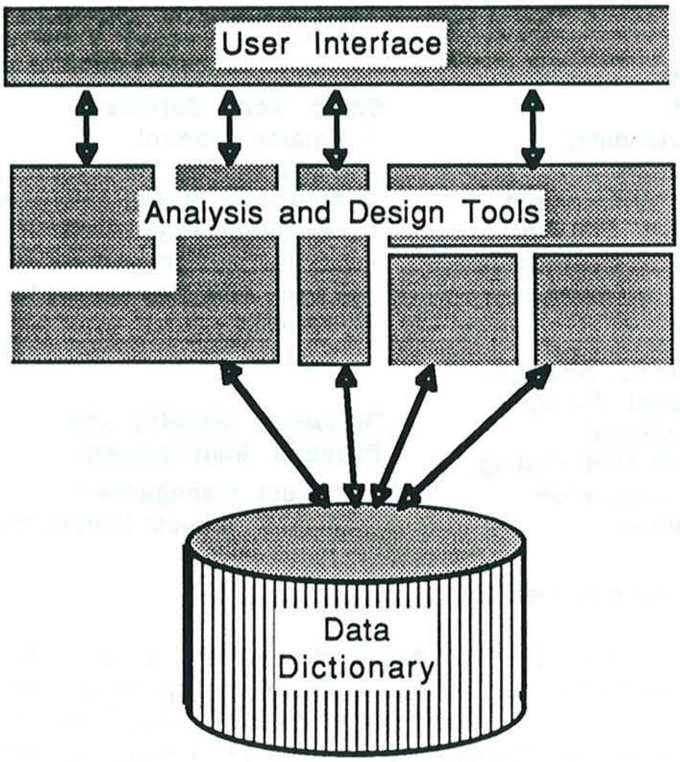
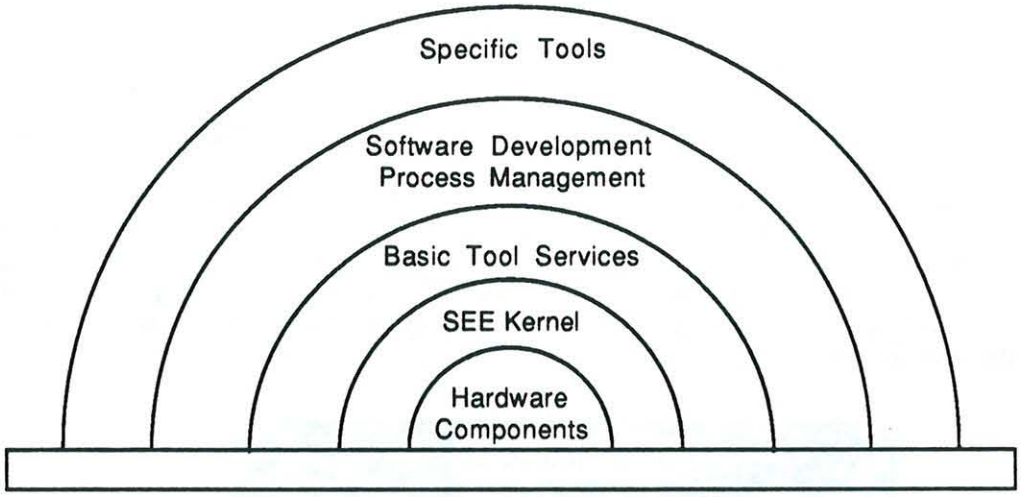


Figure 7 Analyst/designer workbench

An integrated CASE environment (ICASE) is a workbench consisting of compatible tools for the various life cycle phases, so that the output from one tool can serve as the input to the next tool, from requirements capture to code generation. The granularity of the ICASE repositories is on the level of modelling constructs.

An IPSE is an Integrated Project Support Environment. The term used in USA is "software engineering environment". An IPSE provides tools for all phases of the life-cycle. The granularity of the design objects tend to be that of a document rather than a more fine-grained approach. The focus is on managing the systems development and the associated design products. First generation IPSEs have mostly separated their project management database from the design product database. Next generation IPSEs are expected to integrate the two. The database manager provides version and configuration control facilities for the IPSE.



SEE Kernel
- Process Handling
- I/O Facilities
- Database Management
- Distribution Support
- User Interface Support

Specific Tools
- Requirements Analysis
- Architectural Design
- Detailed Design
- Coding and Unit Testing
- System Integration
- Maintenance

Basic Tool Services
- Version Control
- Configuration Management
- Automatic System Building
- Document Preparation
- Document Management
- Basic Data Structure Manipulation
- Security
- Host-Target Communication

Software Development Process Management
- Project Management
- Change Request Management

Figure 8 IPSE architecture

IBM's AD/Cycle and Repository Manager is so far the most ambitious framework that has been proposed by a vendor. AD/Cycle has the structure of an application platform for application system development. The repository manager is at the heart of the system. "A repository is a database of specifications". CASE tools are viewed as add-ons in the platform architecture, and may be acquired from third-party suppliers.

The weak point in todays situation is found in the lack of a comprehensive system specification model. The consequense of this deficiency is that the CASE-tools will remain incompatible for the forseeable future. The next section will discuss possibilities for model integration.

Independently of the quality of the available CASE tools, it seems that the overall quality of the new systems development environments is now so high that we may expect that they will replace the older, conventional environments during the next 10-15 year period. Which consequenses, if any at all, should this have for our curricula? Fairly few of our graduates will be involved in platform design. Some of them will be involved in CASE-tool design, probably in an add-on fashion to some platform. Nearly all of them shall have to do all of their development work within the new environments.

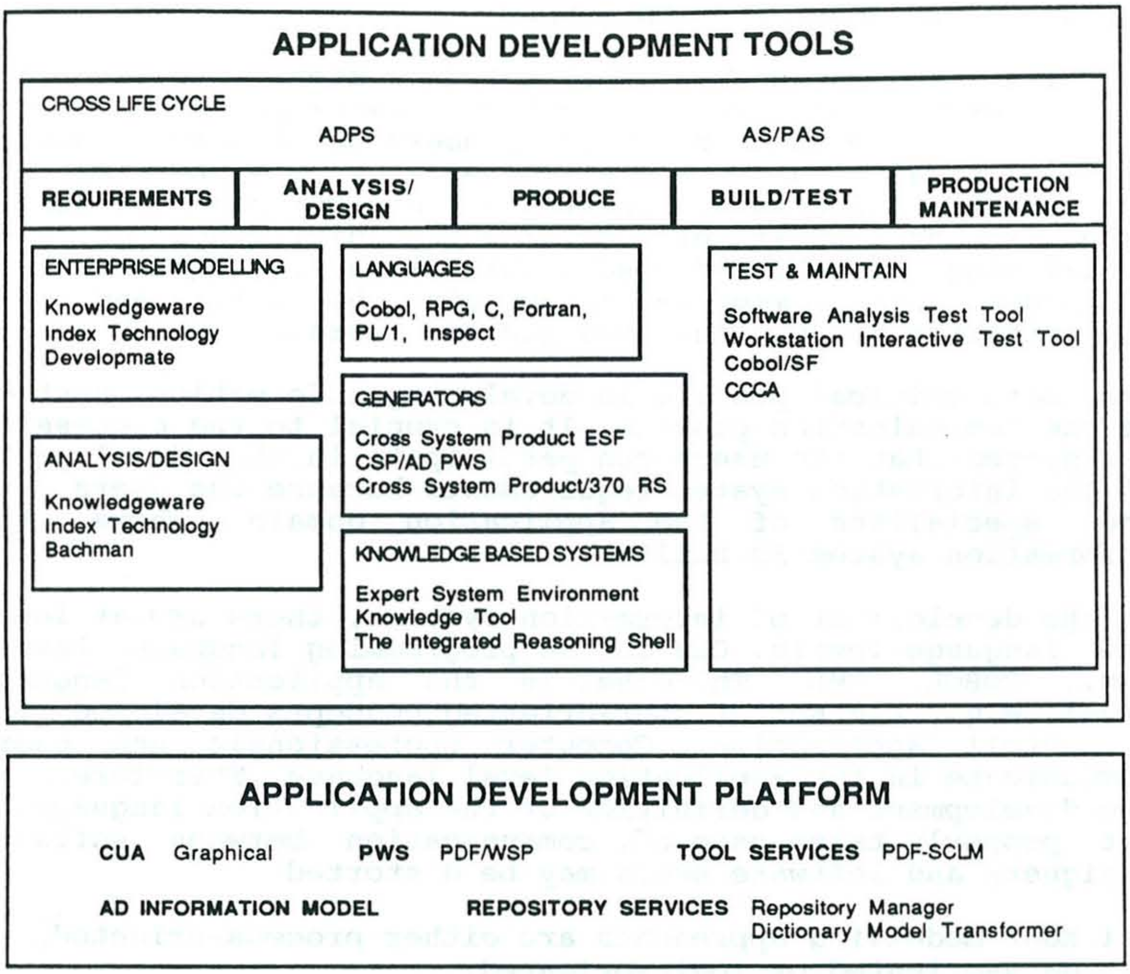


Figure 9 AD/Cycle framework

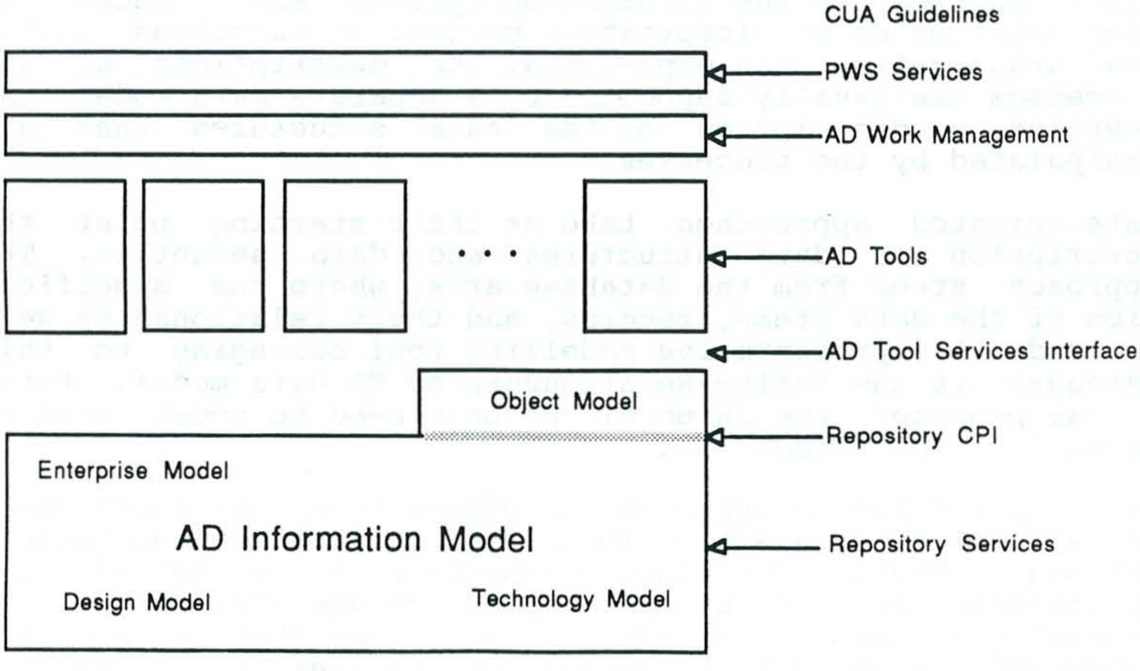


Figure 10 AD/Cycle platform architecture

3. INTEGRATED SPECIFICATION MODELS

To bring large and complex information systems into existence is a team effort involving both the development team and various user communities. In the process one has to consider the political and economic interests of the parties. In particular, the information systems requirements documentation is the result of coordinating differing, or even conflicting opinions and needs. This necessitates the use of languages for communication within the team, and for communication between the team and the users.

The most critical problem in developing information systems is the communication problem. It is crucial to the success of the system that the users can participate in the development of the information system requirements because the users are the specialists of the application domain where the information system is built.

In the development of information systems, there are at least two language levels. One is the programming language level, e.g., COBOL, DML. The other is the application language level, e.g., the set of user-oriented concepts developed for a payroll application. Computer professionals and users communicate in the application level language. Therefore, if the development and definition of the application language is not properly taken care of, communication between software designers and software users may be distorted.

3.1 Most modelling approaches are either process-oriented, data-oriented or logic-oriented.

Process-oriented approaches take as a starting point the description of the processes of the system. This approach was first adopted in the information systems area, where the specification of the information processing functions within the organization was emphasized. The descriptions of the processes are usually supported by a separate data model that contains a description of the data structures that are manipulated by the processes.

Data-oriented approaches take as their starting point the description of data structures and data semantics. The approach stems from the database area, where the specification of the data items, records, and their relationships were focused. A representative modelling tool belonging to this category is the Entity-Relationship or ER data model. After it was proposed, the ER model has been used to model reality as well as modelling data.

In a logic-oriented approach, knowledge about the application is represented by a set of formal assertions, such as logical formulae. These assertions constitute a theory of the application in much the same way as the specification of a deductive database. Events and operations that affect the knowledge base are specified as derivation rules. A derivation rule states that when a set of conditions is true, then its consequences must also be true. That is, it is specied WHAT is to be done when the conditions are true. In this sense, a logic-oriented approach aims at specifying WHAT the informationsystem is going to do rather than HOW to do it. In the processoriented approach, the emphasis is on HOW the information system is to process the information rather than WHAT it is supposed to do. Therefore, the logic-oriented approach has a higher degree of data processing independence than the two other approaches. However, logic-oriented approaches are more difficult to implement and the performance is usually low.

3.2 Modelling approaches may be classified in the temporal dimension

In information system modelling one may divide the methods into static, dynamic, temporal and full time-perspective approaches.

A static model describes only a snapshot of an application problem. A static modelling approach results in a data model e.g. an entity relationship model of teachers, students and courses. It does not reflect the evolution of the application such as the assignment of courses to teachers. In a static model, it is assumed that the events of assigning courses to teachers are considered outside the model. Static models are adequate for applications involving many complex objects and relationships and a small number of events and operations.

Static models are easy to construct, understand, and check. However, many applications require considering the dynamic aspect of applications,where the transition from one state of the system to another needs to be modelled. For example, a retail company information system supporting automated replenishment of parts may require modelling the transactions which affect the stock levels of parts. In some applications, e.g., office information systems, temporal properties involving sequences of states are required to be modelled as well as time points and intervals.

Static approaches provide facilities for describing only a snapshot of the application. Variants of this type may include process models which can be interpreted as computer instructions. The imperative style implies a prescription for the software design. In this approach only one state of reality is explicitly considered at a time. Static approaches were proposed and focused by the mid-1970's.

Dynamic approaches provide facilities for modelling the state transitions without considering the mechanisms that achieve them in full detail. For example, an event or operation can be specified by using a precondition and a postcondition. When a system state satisfies the precondition, the event can take place, or the operation can be performed. In the resulting state the postcondition is true. In this approach, two states are explictly considered at a time i.e., the prestate and the poststate. Dynamic approaches started to be investigated during the late 1970's.

Temporal approaches allow the specification of time dependent constraints such as 'age must not decrease', etc.. In general, sequences of states are explicitly considered in this type of approach. Temporal approaches started to be investigated in the 1980's.

Full time perspective approaches emphasize the important role and particular treatment of time in modelling. A full-time perspective approach eliminates notions such as states, operations, processes, transactions, etc.. The number of states that are explicitly considered at a point in time is infinite. This approach was also introduced in the 1980's.

3.3 Specification languages

A specification language is a structure of modelling constructs which is used for specifying the properties of the system that is being designed. Important roles of a system specification are:

. to serve as a common reference frame for communication among a system's developers

. to serve as a model of reality, offering insight into the application domain

. to serve as a basis for validation and evaluation

. to serve as a basis for implementation

. to provide documentation in order to facilitate system modifications and enhancements

Specification languages may be informal e.g. natural language, or they may be formal e.g. mathematical languages. Most contemporary information systems specification languages are informal, e.g. diagrammatic languages used for sketching system relationships.

The informal specification languages are mainly used as communication tools for systems developers discussing about the functional properties of their systems. Formal languages are mostly used on a rather detailed modelling level, if they are used in practice at all.

We have previously stated that the communication issue is the single most important issue when developing information systems. Specification languages which do not support communication about systems issues among systems developers do not stand any chance at all of being accepted as important development tools. Effective communication can only take place if what is communicated can be understood. The issues must therefore be simplified as much as possible. Unnecessary detail must be abstracted away.

Contemporary specification languages are supporting the simplification process through specialisation. The languages are tailored to the appropriate abstraction levels during the development lifecycle. Dataflow diagrams are used during the functional design, call trees or structure digrams are used during for software module design, and programming languages, decision tables etc. are used during the detailed design of processing rules.

New specification languages are proposed for every new purpose that requires some particular set of parameters to describe the system in their particular view. One example is models and specification languages for performance evaluation. Because performance evaluation tools need to have access to particular system parameters, the system has to be modelled with this particular purpose in mind. Models which have been developed for other purposes can not be used, as long as the language specialisation strategy is used.

3.4 Can a new, high level specification language be developed?

Specification languages with automatic abstraction facilities would improve the current situation. A "super language" would contain sufficient modelling constructs for the specification of every conceivable system detail. In addition, abstraction facilities would be available, so that e.g. a data flow diagram could be derived from the detailed algorithmic specification, plus a specification of the systems structure. This situation is clearly beyond the current state of the art.

Automatic abstraction can be achieved to some limited extent. We have been able to demonstrate abstraction by extending a modified Petri net model with pre- and post-conditions specified in 1st order logic. Pre- and post-conditions for the net as a whole may be derived formally. So the net is replaced by a transition on a higher level of abstraction, with appropriate pre- and post-conditions. For this model we have thus achieved the constructivity property with respect to pre- and post-conditions. This net model is called a behaviour net model (BNM), because some aspects of system behaviour may be specified. BNM specifications are executable in the sense that they may be automatically translated to Prolog code, and may therefore be regarded as a rapid functional prototyper.

To some extent we have been able to adapt the same technique to a specification language which has communicational properties similar to dataflow diagrams. In fact, conventional dataflow diagrams may be automatically abstracted from our specification language. In addition, we are able to automatically produce industrial code (ADA-code) from the specifications. We have developed a solution to C-code generation, but have so far no implementation.

We have still not been able to find a satisfactory solution to the specification of detailed processing rules. So far we either have to formulate ourselves in a logic style, or in a datamanipulation style. We are not satisfied with this, but have so far found no good solution to rule-formulation.

This and other developments of specification languages indicate that a "super-language" may be developed over the next decade. Such a language will combine logic-orientation with process-oriented and data-oriented approaches. The key concept of a "super language" is "constructivity", that is, it must be possible to formally derive the external properties of a system structure when the properties of its components are known, so that the system structure can be replaced by a "black box" with the properties of that system.

DISCUSSION

Rapporteur: Michael Elphick

Lecture One

Professor Gelenbe observed that as the automation of development processes increased, so the notion of a "finished program" became more vague. The result would be that more careful attention should be paid to version management in future. The speaker agreed that, although the platforms provided some forms of version management, this was a very large, important and difficult area. Although sufficient mechanisms were available, he felt that it was particularly important to consider the choice of granularity of version control. They had looked at the control of engineering drawing control in the North Sea oil industry, where it was typical for a new version to be required in about two weeks, and as many as 15 companies might be involved in the process.

Professor Shepherd asked whether it was Dr. Solvberg's view that the availability of such development platforms meant that there would be a lesser role for "traditional" computer scientists in future? The speaker felt that the requirement might be less relatively (but not in absolute terms), although with the development of larger systems, the performance evaluation aspects would become a problem; such analysis needed to be possible at higher levels of abstraction than at present.

Professor Girault raised the question of the availability of tools for ensuring the correctness of synchronisation and coherence of objects. In response, the speaker said that there was a lack of sufficiently good models for this at present, and that building more complex systems would stretch their ability to the limit. There was a lot to learn in this area.

The question of the possible takeover of Software Engineering by these application platforms and the proper role of Computing Science teaching provoked a number of comments: Professor Gelenbe felt that Computing Science should keep to the fundamental topics, while Professor Shepherd commented on the lack of communication between specialists in areas which overlapped in fact. Dr. Solvberg argued that it was not in fact feasible for application system development to be carried out completely by civil engineers, for example, and that Computing Science students should be taught how to cooperate rather than compete.

Finally, Professor Randell drew an analogy between his negative reactions to the increasing complexity and incoherence of this area and similar reactions by many to the introduction by IBM of the System/360 architecture. The subsequent development of more competitive and evolutionary systems (Unix, etc.) was encouraging and a counter to the influence of monetary power. Was there anything similar to hope for in this area? Dr. Solvberg replied that the present lack of integration in modelling would mean that these environments would not be as effective as their designers hoped, but that there would be a gradual improvement as better modelling constructs were developed.

Lecture Two
Professor Girault commented that several different types of diagrams had been presented, and asked whether the non-uniform interface would not be confusing for the user. Dr. Solvberg said that these often referred to different levels of description, and that some (such as the first one shown) could be disastrous for some users. Professor Girault went on to ask about the type of validation analysis performed; this was really complicated in many cases (such as the FIFO ordering in a Petri net analysis, implied by the use of files in a system). The speaker agreed that this was difficult and (at present) not always possible: there was a need to find reasonable limits to what can be done.

After Dr. Sorensen had asked why C had been thought desirable, when the use of Ada seemed more satisfactory, Professor Randell referred to more structured extensions of languages like C (for example C + +). Dr. Solvberg noted that the use of C as a target language required a structural transformation of the specifications; however, they were satisfied with the use of Ada for code generation at present. Finally the speaker mentioned that they had been involved in a couple of ESPRIT projects, with some work in progress involving the use of temporal analysis.