NEAR EAST UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

TABLE OF CONTENTS

INTRODUCTION

Over the past few years, much attention has centred on the need for manufacturing companies within the UK to regain their competitive edge. The international marketplace is changing rapidly, with customers becoming much more demanding with respect to quality and service. British manufacturers have to evolve into more robust and responsive organisations. The CBI National Manufacturing Could determined that UK manufacturing industry has to improve performance by between 20% and 40% in order to achieve internationally competitive standards [1, 2].

Computer-based technology for planning and control has been recognised as a key to competitive manufacturing, providing the right information to the right people at the right time. Information is now recognised as a vital corporate resource, and software is required to support the integration and dissemination of information across the whole manufacturing organisation; information Systems are strategically important within manufacturing. Indeed, IBM has coined the phrase 'information driven manufacturing'. Computer-integrated manufacturing (CIM) has emerged over the past few years in an attempt to define the effective utilisation of manufacturing information across a factory, but today this integration is sill inter-prettied In many ways.

There is a growing need for appropriate computer solutions that can support the business strategy of a manufacturing organisation and enable integration, in an inexpensive yet effective manner A large number of soft-ware vendors are offering disparate packaged systems in support of manufacturing. However, it is often difficult to justify the implementation of manufacturing information technology, particularly within the throes of economic recession. The associated costs are easy to measure on a financial basis, but the benefits are hard to quantify and are often intangible. In order to influence senior management, information technology has to be seen to address specific Identified strategic needs.

A recent survey by Benchmark Research [3] reveals that investment in manufacturing information technology will rise for the first time since 1989, by 2.4% or £33m over the next year within the UK, representing an annual expenditure of £1450 million. The report identifies a low level of satisfaction with information Systems that have been implemented in support of manufacturing planning and control. This is due to the unsuitability of the systems and to the failure to deliver the promised benefits. In response to this report, Puttick [4] stated

There seems to *be a great gulf between IT suppliers and manufacturers, Manufacturers are so caught up in the chaotic world of the factory floor that they are unable to define their problems adequately, let alone explain them to others. IT suppliers, on the other hand, do not fully understand manufacturing, and are so sales-oriented that they don't focus on the real problem.'*

With the expected rise in investment and the proliferation of packaged systems, it becomes increasingly important that manufacturing systems engineers develop a detailed understanding of their organisation's requirements and the problem areas to be addressed. This provides the manufacturing organisation 'with the knowledge necessary to communicate effectively with software vendors and to select the appropriate solutions. Typically, such solutions will be achieved by the selection and tailoring of a range of packages, rather than by the design of bespoke software.

The Advanced Manufacturing Systems Research Group at the University of Liverpool has been researching the specification and design of integrated information systems to support advanced manufacture for a number of years, supported by three major contracts from the ACME (Application of Computers to Manufacturing Engineering) Directorate of the SERC. At an early stage in the Group's work, the importance of a clearly understood and unambiguous requirements specification document was appreciated. Such a document should form the basis of any contract between the manufacturing organisation and a software vendor. To produce such a specification for the complex systems necessary to support an advanced factory is a daunting task

Traditionally, requirements specification documents for manufacturing information systems have been written in natural language and often portray a 'wish-list' predominantly based on the features of commercially available packaged solutions, rather than the strategic needs of the business.

The complexity of the manufacturing environment dictates the need for a systematic or structured approach to enable manufacturing Systems engineers to analyse and define their information requirements. However, very little advice been forthcoming regarding this procedure within manufacturing industry. Many authors have identified the need for structured approaches and effective systems modelling techniques, but the few methods developed to satisfy this need have failed to find 'widespread use.

Within the comparatively youthful computer industry, the discipline of software engineering and computer-aided software engineering (CASE) tools have been proven to facilitate the production of reliable, 'well documented information systems that help meet the requirements of the intended users. That is due to the considerable emphasis on the tasks of requirements specification. However, almost ah of this work has taken place outside the manufacturing environment.

If as research at the University of Liverpool suggests, manufacturing Systems engineers are required to analyse their information difficulties and define the associated requirements, it would seem appropriate to utilise established approaches from other application domains, such as the software industry. Although this route is logical and feasible, the methods and tools available for use by the non-specialist (in software engineering terms) manufacturing systems engineer need much development.

This paper presents the research 'work undertaken to Investigate the relevance of software engineering and CASE in manufacturing systems engineering. An in-depth examination of this relevance had not previously been undertaken, and the indications from the manufacturing industry are that CASE has had little real penetration. The research objectives were

- to establish the need for structured methods and tools in support of the production of robust requirements specifications for manufacturing information systems.

- to examine commercially available CASE methods and tools for their suitability to provide support to the manufacturing systems engineer in developing a specification of requirements for manufacturing information systems.

- to identify the main developments needed in CASE methods and tools to ensure their widespread adoption within manufacturing industry; this formed the basis of the contribution to new knowledge.

An extensive review of literature was conducted into manufacturing information Systems development and the history and current capabilities of CASE methods and tools. In addition, this review examined the tentative use of CASE methods and tools within the manufacturing industry. A survey was undertaken of commercially

available CASE products aimed at supporting the specification of requirements to increase familiarisation 'with the CASE marketplace and product capabilities. Three typical CASE products, Identified as suitable, were applied on-site to manufacturing information systems projects, This tested the appropriateness of the methods and tools employed within the manufacturing environment.

In addition, experts within manufacturing information Systems development (from both academia and industry) were questioned to gain an external perspective of the needs of manufacturing systems engineering and the relevance of CASE methods and tools. The Initial research hypotheses were examined in the light of the research work undertaken, and conclusions and recommendations were produced for the requirements of CASE for more widespread application within manufacturing industry. It is important to note that this research project was viewed as an introductory study into this area, raising many questions to be tackled in a subsequent project outlined.

We believe that, in order to facilitate a competent definition of a manufacturing organisation's requirements for computer-based technology in support of its strategic objectives, it is necessary for manufacturing systems engineers to adopt structured approaches and tools. As Puttick [5] states.

'This mismatch between the Systems manufacturing has installed and what It really needs IS due to an Inability of manufacturing to articulate its needs and a Lack of understanding by the IT vendors... The lack of analysis toots and techniques to translate business and manufacturing needs into if requirements has held industry back.'

The complexity of information Systems necessitates the need for a methodical approach to their development Over the years, this need has resulted in a wide range of methodologies being developed to support differing development approaches. Despite the availability of such methodical approaches, the design of information systems remains largely a knowledge-Intensive activity, beginning with an informal set of frequently vague requirements and ending up with a systematically defined formal object [1]. Systems design has been described as being a labour intensive process, much prone to error, with the end result of the design process being devoid of the design know-ledge that led to its construction [1]. Although contemporary computer-aided software engineering (CASE) tools provide assistance In carrying out many design tasks with improved efficiency, they are largely the results of the

automation of established design techniques. Often, existing CASE tools are little more than graphical front-ends to data dictionaries [2]. In general, the fundamental characteristic of design Is not addressed by existing CASE technology.

INTRODUCTION

The complexity of information Systems necessitates the need for a methodical approach to their development Over the years, this need has resulted in a wide range of methodologies being developed to support differing development approaches. Despite the availability of such methodical approaches, the design of information systems remains largely a knowledge-Intensive activity, beginning with an informal set of frequently vague requirements and ending up with a systematically defined formal object [1]. Systems design has been described as being a labour intensive process, much prone to error, with the end result of the design process being devoid of the design know-ledge that led to its construction [1]. Although contemporary computer-aided software engineering (CASE) tools provide assistance In carrying out many design tasks with improved efficiency, they are largely the results of the automation of established design techniques. Often, existing CASE tools are little more than graphical front-ends to data dictionaries [2]. In general, the fundamental characteristic of design Is not addressed by existing CASE technology.

Knowledge-based CASE tools

Artificial intelligence (AI) technology can be used to develop so called knowledge-based CASE tools (KB-CASE).

Although a small number of existing CASE tools have limited knowledge-based capabilities. In terms of checking graphical correctness of analysis and design models or applying the rules or a particular methodology to the models created by the designer, the real promise of such tools lies elsewhere [3]. Rather than simply commenting on and or validating a model that a human has constructed, KB-CASE tools (sometimes referred to as *design agents)* are able to play an active part during the design process. They are capable of providing Intelligent assistance when required In the form of advice, suggesting alternative solutions, helping to investigate the consequences of design decisions, and maintaining the availability of the design knowledge by providing information should a design decision be questioned or require explanation in retrospect. They are an attempt to maintain the availability of the knowledge used during the design after the initial design process has been completed [4]. Such tools have an understanding of both the structure and the semantics of the design [1].

Several researchers have previously applied AI technology to develop KB-CASE tools Such tools have been developed to assist at all stages of the design process, and are generally classed as those supporting specification acquisition (including conceptual, logical and physical design) and those supporting program

synthesis for tools for specification acquisition are concerned with the generation of complete, consistent and correct design specifications, and/or their validation and evaluation. Such specifications generally describe the user's requirements from the system, the structure of the system (in terms of software modules and flow of control) and the structure of the data to be used.

According to Gero and Maher [6], Innovative (or novel) design occurs when designs. In the search space, which have not been produced previously, are generated or found. Creative design occurs when designs are discovered that did not exist conceptually before the system (i.e. the extension of the boundaries of the search space). Given these definitions, KB-CASE tools supporting specification acquisition can generally be thought of as performing innovative rather than creative design. Tools for program synthesis support the transformation of designs into executable code by attempting to synthesise efficient programs from initial program specifications. This is generally achieved by the gradual refining of a high-level specification until a program satisfying the original specification is obtained. This work concentrates on tools supporting specification acquisition.

Given that artificial intelligence and conceptual modelling have developed similar knowledge representation formalism's [7], it is not surprising that a number of researchers have previously applied Al to develop KB-CASE tools. What is surprising is the relative lack of activity in an area of such potential. Examples of interest illustrating previous works on tools supporting specification acquisition Include The Programmer's Apprentice [8], SECSI [9], VCS [10], and Modeller [11]. The Programmer's Apprentice is designed to provide support at various stages of the development process. In this sense, its aims are similar to those of Modeller, an integral part of a family of intelligent products being developed by COGNOS Inc., to assist In the process of systems design. The code specification module of both these projects can be classed as performing program synthesis. SECS provides intelligent support for logical database design and is widely seen as the seminal work in the area of intelligent database design tools, having Influenced a number of successors (including VCS) and made a commercial break-through in being marketed by infosys.

Areas for Improvement

A framework for the evaluation of KB-CASE tools supporting specification acquisition derived from an Investigation into the state of development of such tools is described elsewhere [12]. A brief description of the

criteria within this framework follows. Table I provides an overview of the results of this review with regard to these criteria.

• *Stage or design covered:* which stage(s) of the chosen design paradigm it attempts to support.

criteria	description
stage of design covered	the majority of tools examined attempted to provide support for a single stage of the chosen design paradigm (such as logical design); very few attempted to provide support for more than one stage of the development process
user interface employed	tools generally provided a menu-based interface, some form of natural language interface (NLI), or a combination of both; minority provided graphical capabilities in addition to some form of menu/NLI combination
method used to drive process	tools generally fell into one of two categories; those requiring continuous input from an intended user of the application system (based on the assumption that the best source of information relating to a target system will in fact be a user of that system), and those relying on systems development staff to analyse the target domain, and to present the information in an appropriate form to the tool
domain-specific knowledge	this criteria was the least well supported of all; despite the potential of increasing the appearance of intelligence of CASE tools, this criteria was almost completely ignored
design technique used	the vast majority of tools provided active support for well established design techniques; few developers strayed from the path of supporting techniques that were already generally accepted
undo facilities	this was obviously seen as a fundamental requirement by most developers as the majority of tools provided some facility for undoing design decisions
learning ability	although a number of tools could 'learn' about the application domain during the course of a design session, few tools could put this evolving knowledge to use
ease of use	the majority of tools examined appeared to be reasonably straightforward to use by the intended operator; some offered particularly friendly interfaces combining graphical and natural language features; a minority, however, imposed prerequisites that would be difficult to meet for many everyday users, such as knowledge of specific techniques and/or specification languages, or the availability of specialist staff

Table 2. Summary of evaluation of intelligent design tools.

• *User interface employed:* the method by which a system receives Information from the user.

• *Method used to drive design process:* the method used to provide initial input to the system Is examined i.e. how initial information relating to the application domain is gathered; supplied directly by the user or provided by some other means. The driving mechanism Is also examined in terms of whether continuous user input is required throughout the design process, or whether the process is largely automatic once initial information has been gathered.

Areas for further Investigation include the transfer of knowledge gained In one design session through to other sessions, i.e. the reuse of knowledge previously gained should it be applicable (currently, such knowledge is generally restricted to within a single design session). Perhaps the area with the greatest potential is also the most neglected; that of domain-specific knowledge. Domain-specific (or 'real world) knowledge and the ability to reason with this knowledge would be of obvious advantage to an intelligent design tool. Successful advances in this area would also appear to have potential in the area of improving acceptability, as tools that appear more Intelligent and efficient would presumably be more acceptable to users.

In the remainder of this paper, we illustrate how domain-specific knowledge may be exploited In order to Improve the performance of KB-CASE tools supporting specification acquisition.

Current generation CASE tools for analysis and design were conceived in the late I 970s and 1980s, largely as vehicles for the automation of structured techniques such as data flow diagrams and entity-relationship modelling [1]. These paper-based techniques incurred heavy administrative overheads in maintaining records, consistency checking, and producing analysis and design deliverable's. CASE was hailed as a solution to the problems of managing and manipulating diagrams and the large data dictionaries, or repositories, needed to record information about diagram objects. The advantages of CASE included the ability to print and reprint diagrams and to perform automatic consistency checking. CASE tools were able to automate the production of analysis and design deliverable's and, more recently, they have offered some ability to produce working software Systems with only limited intervention by programmers. The growing number of IS methodologies made use of the same diagrammatic techniques; CASE was seen as a way of enforcing the use of these techniques, which were generally perceived as 'better' than past practices.

Anyone familiar with contemporary management research, or involved in the IT industry, will have heard about business process re-engineering (BPR). In BPR, the concept of IT as a way of simply automating existing business activities is discarded in favour of a far more Interventionist style. it is well known that many new computer-based Information systems fail to produce hoped-for Improvements In productivity, levels of service, and so on. This failure is often blamed on the fad that these new Systems attempt to automate existing 'bad' practices. BPR aims to attain true improvement through first examining current ways of working, and then improving those practices by redesigning work flow, restructuring departments, tuning procedures etc. The application of IT often goes hand-in-hand with this process redesign; indeed, the new ways of working

may well be Impossible without ft. Hammer and Champy [2] point out the special importance of IT as a *disruptive* technology, with 'its ability to break the rules that limit how we conduct our work' The current focus on IT as a means of attaining competitive advantage hinges on the same kind of ideas as are embodied in the BPR approach [3].

Systems analysis and design: a case for BPR?

The discipline of systems analysis has evolved through several stages. When first conceived, several decades ago, systems analysis was primarily a general problem-solving approach; with the Introduction of computers, It came to be linked more closely to the design of computer-based information Systems. The use of analytical tools has always been emphasised, but the level of formality has changed; highly 'structured' specification techniques, such as object-oriented analysis, are often now used where, in the past, prose specifications were considered adequate [4].

Before the advent of 'structured' methods, a common approach by systems analysts to the design of new computer-based information systems would be to construct a 'user requirements statement', a document in which prose descriptions of requirements for the new computer system would be interspersed 'with hand-drawn report and screen layouts. Flow charts could be used to represent the processing to be performed by the new system. File layouts might be included to show the data items to be stored. One of the major problems of documents of this type 'was Inaccessibility; the sheer size and complexity was Intimidating. Such statements of requirements "were (and, in fact, still are) often used as the basis for contractual arrangements. The use of 'natural' language meant that anyone could read the documents. However, it was difficult to tell if they were coned or internally consistent, and few formal means of checking were available.

What we have seen since then is an explosion in the number of structured techniques for representing requirements. enough prose Is still used, especially at the earliest stages of requirements analysis, the focus is on semi-formal, and often diagrammatic, representations. Techniques such as entity-relationship modelling, state-transition diagrams, data flow diagrams, function hierarchy charts, and matrices of all descriptions are used to codify and formulate requirements. Contemporary information systems methodologies, such as SSADM, place great weight on the correct application of these techniques, and their use is considered to be 'good' In comparison with less structured approaches. Fig. XXX is an example of a diagram used In one type of 'structured' analysis technique, object-oriented analysis [5].

Following the dictates of the market-place, most CASE tools for analysis and design support the creation and maintenance of entity-relationship diagrams, data flow diagrams etc. Unfortunately, in doing so, they may have made the same mistake that many early Systems developers made when designing new business application systems; the automation of existing practices, with insufficient thought given to process redesign. The result now is that many CASE tools can capture the diagrams and supporting data produced by common Systems analysis techniques. However, they do not support the way in *which those techniques are applied.* Below, we explore this issue in greater depth.

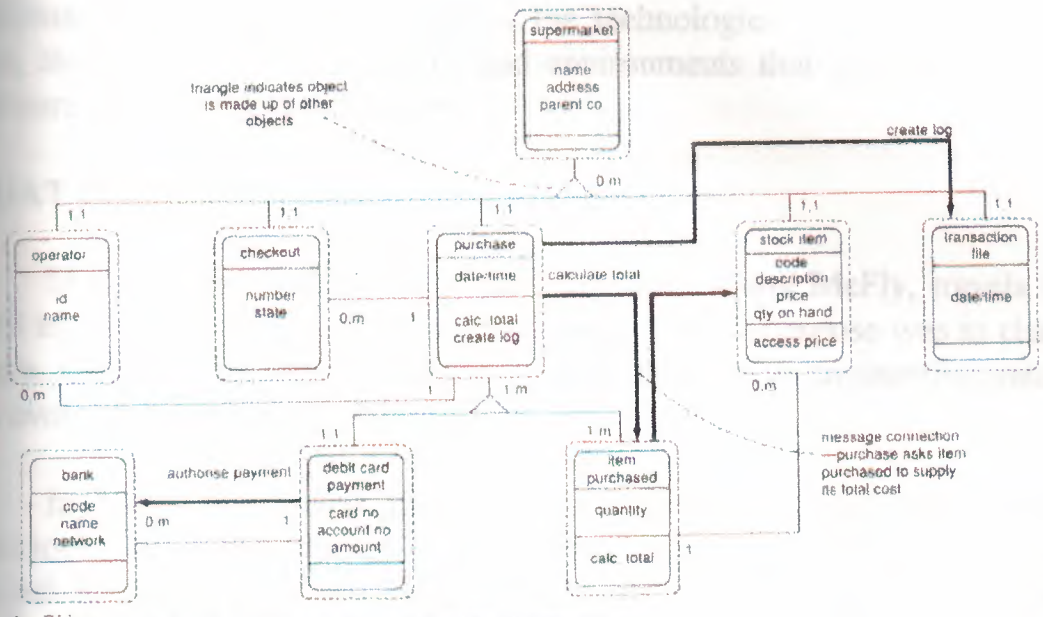


Figure 38. Object-oriented analysis diagram.

COMPUTER-AIDED SOFTWARE ENGINNERING (CASE)

Everyone has heard the saying about the shoemaker's children: The shoemaker is a so busy making shoe for others that his children don't have shoes of their own. Over the pest 20 years, many software engineers have been the "shoemaker's children." Although these technical professionals have built complex systems that automate the work of others, they have used very little automation themselves. In fact, until recently software engineering was fundamentally a manual activity in which tools were used only at the latter stages of the process.

Today, software engineers have finally been given their first new pair of shoes-computer-aided software engineering (CASE). The shoes don't come in as many varieties as we would like, are often a bit stiff and sometimes uncomfortable, don't provide enough sophistication for those who are stylish and don't always match other garments that software developers use. But they provide an absolutely essential piece of apparel for the software developer's wardrobe, and will; over time, become more comfortable, more useable, and more adaptable to the needs of individual practitioners.

In this chapter, the technical aspects of computer-aided software engineering are discussed. CASE technologies span a wide range of topics that encompass software engineering methods and project management procedures. In earlier chapters of this book we have attempted to provide a reasonable understanding of the underpinning of the technologies. In this chapter and the next, the focus shifts to the tools and environments that will help to automate software engineering technologies.

WHAT IS CASE?

In the movie Back to the Future, the hero, Marty McFly, travels back to 1955 in a souped-up DeLorean time machine. Marty's purpose was in change his future. Ours will be more mundane: to understand how engineering automation has evolved over the past 40 years.

In 1955, mechanical and electrical engineers worked with rudimentary hand tools books and tables that contained the formulae and algorithms that they needed for analysis of an engineering problem: slide rules and calculators mechanical, not electronic!) for doing the computation necessary to ensure that the product would work; pens and pencils, drafting boards, rules, and other paraphernalia that enabled the engineer to create models of the product that was to be built. Good work was done, but it was done by hand.

A decade passed and the same engineering group begun experimenting with computer-based engineering. Many staff members resisted the use of computers. "I just don't trust the results," was a common complaint But many others jumped in with both feet. The engineering process was changing "

We jump to 1975. The formulae and algorithms that the engineers needed were embedded in a large suite of computer programs that were used to analyse a wide array of engineering problems. People trusted the results of these programs. In fact, much of their work could not be accomplished without them. Computer

graphics workstations, tied to large mainframes ,were in use in a few commands and had replaced the drafting board and related tool for the creation of engineering models. A bridge between engineering and manufacturing work was under construction creating the first link between computer-aided design (CAD) and computer-aided manufacturing (CAM).

Good work continued to be done, but it was now dependent on software. Computing and engineering had been joined inextricably.

Arriving back at the present, we see computer-aided engineering (CAE) computer-aided design, and computer-integrated manufacturing (CIM the successor to CAM) as commonplace activities in most companies. Engineering automation has not only arrived, it is an integral part of the process.

Unlike Marty McFly, mechanical and electrical engineers can't go back and change the future. But, in a way, software engineers can. They have the opportunity to mold the future of CASE by learning lessons from the evolution of CAE, CAD, and CIM.

A Software Engineering Workshop

The best workshops have three primary characteristics: (1) a collection of useful tools that will help in every step of building a product; (2) an organised layout that enable the tools to be found quickly and used efficiently; (3) a skilled crafts person who understands how to use the tools in an effective manner. Software engineers now recognise that they need more and varied tools (hand tools alone just won't meet the demand, of modern computer based systems). They also need an organised and efficient workshop in which to place the tools.

The workshop for software engineering is called an integrated project support environment and the tool sell that fills the workshop is CASE.

An Analogy

It is fair to state that computer-aided software engineering has the potential to become the most implant technological advance in the history of software development. The key word in the preceding sentence is "potential".

Today, CASE tools add to the software engineers tool box. CASE provides the engineer with the ability to automate manual activities and to improve engineering insight. Yet to become "the most important technological

advance" CASE must do much more. It must from the building block of a workshop for software development.

Today, CASE is where CAD/CAE/CIM were in 1975 individual tools are being used by some companies, usage across the industry is speeding rapidly. and serious effort is under-way to integrate the individuals tools to form a consistent environment.

There is little doubt that CASE will impact software engineering in substantially the same way that CAE/CAD/CIM has impacted other engineering disciplines. However there are some important differences. During its early years of evolution CAD/CAE/CIM implemented engineering practices that had been tried and proven over the past 100 years. CASE, on the other hand, provides a set of semi-automated and automated tools that are implementing an engineering culture that is new to many companies. The difference in impact and in acceptance is profound.

CAD/CAE focuses almost exclusively on problem solving and design, It continues to struggle with a bridge to manufacturing through CIM. The primary goal of CASE (over the long haul) is to move toward the automation.

BUILDING BLOCKS FOR CASE

Computer-aided software engineering can be as simple as a single tool that supports a specific software engineering activity or as complex as a complete "environment" that encompasses tools, a database, people, hardware, network, operating systems, standard, and myriad other component. In this section, an overview of the building blocks that create a CASE environment is presented. These building blocks are discussed in the context of CASE environments.

The building blocks for CASE are illustrated in Figure 1. Each building block forms a foundation for the next, with tools sitting at the top of the heap. It is interesting to note that the foundation for effective CASE environments has relatively little to do with software engineering tools themselves. Rather, successful environments for software engineering are built on an environment architecture that encompasses appropriate hardware and systems software. In addition, the environment architecture must consider the human work patterns that are applied during the software engineering process.

During the 1960s, 1980s software development was a mainframe activity. Terminals ware linked to a central computer and each software developer shared the resource of that computer. Software tools that were available (and there were

relatively few) were designed to operate in a terminal-based time-sharing environment.

Today the trend in software development is away from the mainframe computer and toward the workstation as a software engineering platform.

Figure 1 . CASE building block.

Individual workstations are networked so that software engineers can communicate effectively. The project database is available through a network file server that is accessible from all workstations. An operating system that supports the hardware, the network, and the tools ties the environment.

The environment architecture, composed of the hardware platform and operating system support (including networking and database management software), lays the groundwork for CASE. But the CASE environment itself demands other building blocks. A set of portability services provides a bridge between CASE tools and their integration framework and the environment architecture. The integration framework is a collection of specialised programs that enables individual CASE tools to communicate with one another to create a project database, and to exhibit the same look and feel to that end user (the software engineer. Portability services allow CASE toots and their integration framework to migrate across different hardware platforms and operating systems without significant adaptive maintenance.

The building blocks depicted in Figure1 represent a comprehensive use today have not been constructed using all of a budding blocks above. In fact, the majority of CASE tools are "point solutions." That is a tool is used to assist in a particular software engineering activity (e,g., analysis modeling), but does not directly communicate with other tools is not tied into a project database and is not part of an integrated CASE (I-CASE) environment. Although this situation is not ideal a CASE tool can be used quite effectively, even if is a point solution.

The relative levels of CASE integration are shown in figure 2. At the low end of the integration spectrum is the individual (point solution) tool. when individual tools provide facilities for data exchange (most do), the integration level is improved slightly. Such tools produce output in a standard format that should be compatible with other tools that can read the format. In some cases, the builders of complementary CASE tools work together form a bridge between the tools (e.g., analysis and design tool that is coupled with a code generator). Using this approach, the synergy between the tools can produce end products that would be difficult to create using either tool separately. Single-source integration occurs when a single CASE tools vendor integrates a number of different tools and sells them as a package. Although this approach is quite effective the closed architecture of most single-source environments precludes easy addition of tools from other vendors.

At the high end of the integration spectrum is the integrated project support environment (IPSE). Standards for each of the building blocks described above are created. CASE tools vendors use these IPSE standard.

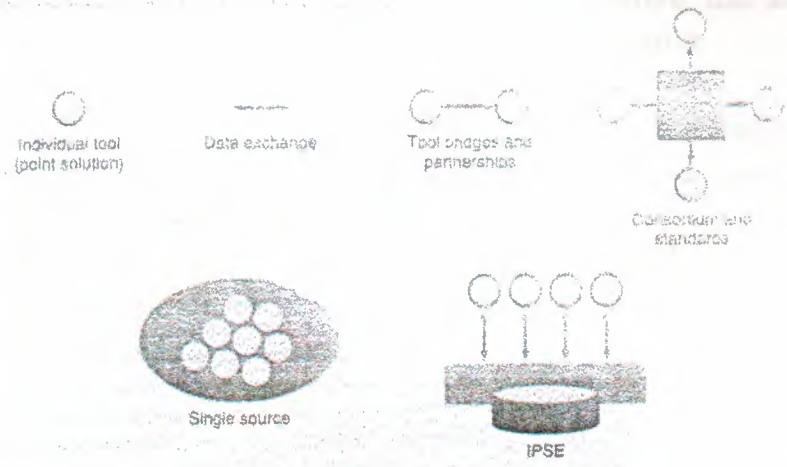


Figure 2. Integration options.

TAXONOMY OF CASE TOOLS

A number of risks are inherent whenever we attempt to categorise CASE tools. There is a subtle implication that to create an effective CASE environment one must implement all categories of tools-but this is simply not true. Confusion (or antagonism) can be created by placing a specific tool within one category when others might believe it belongs in another category. Some readers may feel that an entire category has been omitted- thereby eliminating an entire set of tools for inclusion in the overall CASE environment. In addition simple categorisation

tends to be flat- that is we do not show the hierarchical interaction of tools or relationships among them. But even with these risks it is necessary to create a taxonomy of CASE tools- to better understand the breadth of CASE and to better appreciate where such tools can be applied in the software engineering process.

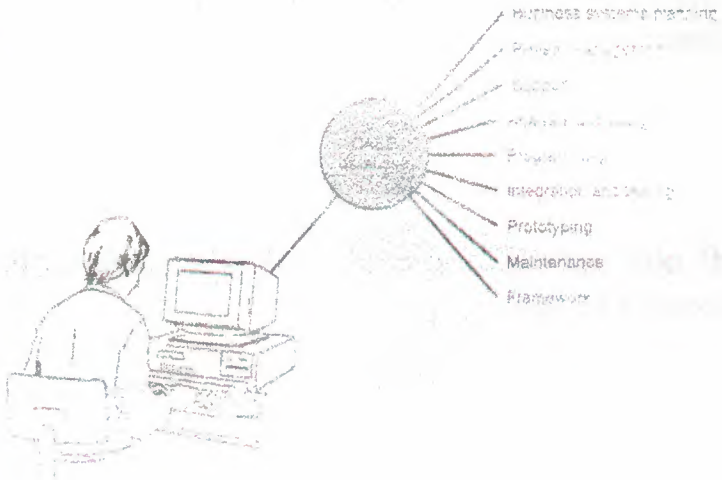
CASE tools can be classified by function, by their role as instruments for managers or technical people, by their use in the various steps of the software engineering process, by the environment architecture (hardware and software) that supports them, or even by their origin or cost (QED89). The taxonomy created in this book (Figure 3) uses function as a primary criteria.

BUSINESS SYSTEMS PLANNING TOOLS

By modeling the strategic information requirements of an organisation. business systems planning tools provide a "meta-model" from which specific information systems are derived. Rather than focusing on the requirements of a specific application, business information is modelled as it moves between various organisational entities within a company [MAR89]. The primary objective for tools in this category is to help improve the understanding of information moves between the various organisational units.

It is important to every organisation. They require a major commitment in resources and a major philosophical commitment by management to produce a complete model and then act upon the information derived from it. However such tools to provide substantial insight when information system strategies are to be constructed systems and methods do not meet the needs of an organisation.

Figure 3. A CA



PROJECT MANAGEMENT TOOLS

Many software project managers continue to estimate, control, and track software projects in much the same way that these activities were performed during the 1950s. Ironically, there is a broad array of CASE project management tools that could have a profound impact on the quality of project management for software development efforts both large and small.

Today, most CASE project management tools focus on one specific element of project management, rather than providing all-encompassing sup port for the management activity. By using a selected set of CASE tools, the "' project manager can generate useful estimates of effort, cost, and duration of a software project, define a work breakdown structure (WBS) and plan a workable project schedule, and track projects on a continuing basis. In addition the manager can use tools to collect metrics that will ultimately provide an indication of software development productivity and product quality. For those managers who have the responsibility for contract software development, CASE tools are available ta trade requirements from the original customer request proposal (RFP) to the software development work that implements these requirements in a deliverable system.

Project Planning Tools

Tools in this category focus on two primary areas software project effort and cost estimation and project scheduling. Cost estimation tools enable the project manager to estimate e.g.. problem complexity, estimated effort, project. recommended number of people using one or more of the techniques intra. Many tools in this category allows some form of game playing. For example, the project manager can permute the project deadline and examine its impact on overall cost.

Project scheduling tools enable the manager to define all project tasks (the work breakdown structure), create a task network (usual using graphical input), represent task interdependencies. and model the amount of parallelism possible for the project. Most tools use the critical-path scheduling method to determine the impact of slippage on delivery date.

Requirements Tracing Tools

When large systems are developed, things "fall into the cracks." This phrase refers to a critical problem in the development of computer-based system does the delivered system does not fully meet customer-specified requirements

arises from technical difficulties. But in other situation the requirements are missed they simply were not addressed.

The objective of requirements tracing tools (Figure 4) is to provide a systematic approach to the isolation of requirements, beginning with the customer RFP or specification. The typical requirements tracing tool combines human interactive test evaluation, with a database management system that is "parsed" from the original RFP or specification categorises each requirement that from the original or specification. The parsing of requirements can be as simple as finding every occurrence of the verb "shall" (indicative of a requirement) and the highlighting the statement, in which "shall" appears. The analyst then categorises the requirement implied by the sentence and enters it into a database. Subsequent development work can be cross-referenced to the database so that conformance to requirements is more likely.

Metrics and Management Tools

Software metrics improve a manager's ability to control and coordinate the software engineering process and a practitioners ability to improve the quality of the software that is produced. Today's metrics or measurement tools focus on process and product characteristics. Management-oriented tools capture project specific metrics (e.g., LOG person-month, defect per function point) that provide an overall indication of productivity or quality . Technically oriented tools determine technical metrics (e.g., cyclomatic complexity) that provide greater insight into the quality of designer or code. Many of the more advanced metric tools maintain a database of industry average measures. Based on project and product characteristic provided by the user, such tools, rate local numbers against for improvement.

Management tools (exclusive of project estimation and scheduling tools) assist information systems managers in prioritizing the many projects that compete for limited development resources. Using customer requirements and priorities constrains placed on the development organisation, and technical and business risks, such tools use an expert system approach to suggest the order in which a project should be undertaken.

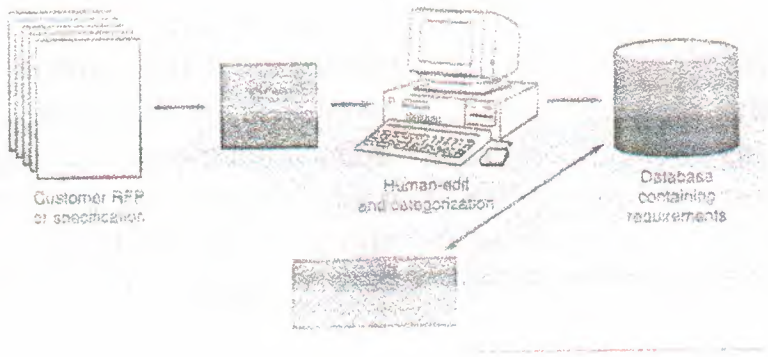


Figure 4. Treatments tracing tools.

SUPPORT TOOLS

The support tools category encompasses systems and application tools that complement the software engineering process. Tools in this broad category. encompass the umbrella activities that are applicable across the entire software engineering process. They include documentation tools, system software and networking tools, quality assurance tools, and software configuration management and database management tools (also members of framework tools category).

Documentation Tools

Document production and desk top publishing tools support nearly every aspect of software engineering and represent a substantial leverage opportunity for all software developers. Most software development organisations. spend a substantial amount of time developing documents, and in many cases the documentation process itself is quite inefficient. It. is not unusual for a software engineering, organisation to spend as much as 20 or 30 percent of all its software development effort on documentation. For this reason documentation tools provide an important opportunity to improve productivity.

Documentation tools are often linked to other CASE tools using a data bridge implemented by the vendor of the technical tool. For example, a number of analysis and design tools have links to one or more desktop publishing systems, so that models and text created during analysis and design can be transmitted to a documentation tool and embedded in the specification created using the documentation tool.

System Software Tools

CASE is a workstation technology. Therefore, the CASE environment must , accommodate high-quality network system software, electronic mail, bulletin boards and other communication capabilities. Although the operating system of reference for most engineering workstations (and an increasing number of high-end PCs) in UNIX, the portability services provided by an IPSE may enable CASE tools to migrate to other operating systems without great frustration.

Quality Assurance Tools

The majority of CASE tools that on quality assurance are actually metrics tools that audit source code to determine compliance with language standards. Other tools extract technical metric (see Section 22.5.3) in an effort to project the quality of the software that is being built.

Database and SCM Tools

Database management software serves as a foundation for the establishment of a CASE database (repository) that we have called the project database. Given the emphasis on configuration objects, database management tools for CASE may evolve from relational database management systems (RDMS) to object-oriented database management systems (OODMS).

Proponents claim that an OODMS (GUP9l) will make configuration management easier is accomplish and argue that the object-oriented structure is a natural organisation for software configuration items that combine many different types of information. Proponents for RDMS claim better performance and significantly more industry experience than OODMS and argue that the relational model easy accomplish most, if not all, of the capability that can be achieved using the object-oriented model. Only time will tell which approach predominates for CASE databases.

CASE tools can assist in all five major SCM tasks-identification, version control, change control, auditing, and status accounting. The CASE database provides a mechanism for identifying each configuration item and relating it to other items; the control process discussed. Can be implemented with the aid of specialised tools easy access to individual configuration item facilitate the auditing process and CASE communication tools can greatly improve status accounting (reporting information about changes to all who need to know).

Software configuration management lies at the kernel of every CASE environment. By controlling changes to the software configuration, SCM tools enforce human cognisance of each change, thereby reducing misunderstanding and improving system quality.

The use of the database, configuration management tools. and specialised "browsing" tools provides a first step toward the creation of a library for software that will encourage the reuse of software components. Although relatively little reuse has been accomplished to date, CASE offers the first real promise for achieving broader reuse of computer software components.

ANALYSIS AND DESIGN TOOLS

Analysis and design tools enable a software engineer to create a model of the system to be built. The model contains a representation of data and control flow data content (through a definition of a requirements dictionary), process representation control specification and a variety of other modelling representations. Analysis and design tools assist in the creation of the model and also in an evaluation of the models quality. By performing consistency and validity checking on the model, analysis and design tools provide a software engineer with some degree of insight into the analysis representation and help to eliminate errors before they propagate into the design, or worse, into implementation itself.

SA/SD Tools

Most analysis and design tools implement the structured analysis and structured design (SA/SD) method discussed. SA/SD is a modelling technique. It enables a software engineer to create progressively more complex models at a system, beginning at the requirements level and finishing with an architectural design. SA/SD combines a specific notation; analysis and design heuristic, and an analysis-to-design transformation process (a mapping) to produce workable representations of software.

PRO/SIM Tools

Phototyping and simulation (PRO/SIM) tools [NIC9O] provide the software engineer with the ability to predict the behaviour of a real-time system prior to the time that it is built. In addition, it enables the software engineer to develop mock-ups of the real-time system that allow the customer to gain insight into the function, operation, and response prior to actual implementation there is

little doubt that such capability provides distinct benefits in an area where success has been unpredictable and software development itself is something of a black art .

Most PRO/SIM tools provide the software engineer with a mean for creating functional and behavioural models of a system. Tool in this category provide a mean for specifying projected performance characteristics of each system element (e.g., execution speed of a hardware or software function) defining the input and output data characteristics (e.g., input data arrivals rates or interrupt characteristics), and modelling the interface / interconnectivity among system elements.

Many PRO/SIM tools provide a code generation capability for Ada and other programming languages that will likely become considerable more sophisticated as new generations of these tools evolve. In addition, all tools in this category make use of an underlying formal or quasi-formal specification language opening the door to more comprehensive code generation formal verification of the system specification.

Interface Design and Development Tools

Even with the evolution of user interface standards, the design and development of human-computer interfaces remain a challenge for software engineers. Industry studies have found that between 50 to 80 percent of all code generated for interactive application is generated to manage and implement the human-computer interface [LEE9O].

Interface design and development tools are actually a tool kit of program components such as menus, buttons, window structures, icons, scrolling mechanisms, device drivers, and so forth. However, these tool kits are being replaced by interface phototyping tools that enable the rapid on screen replaced by interface creation of sophisticated user interfaces that conform to the interlacing standard (e.g. X-Windows, Motif) that has been adopted for the software.

User interface development systems (UIDS) combine individual CASE tools for human computer interaction with a program components library that enables a developer to build a human-computer interface quickly [MYE89]. A UIDS provides program components that manager input devices, validate user inputs, handle error condition process and "undos," provide visual feedback. prompts, and help, update the display manage application data, handle scrolling and editing, insulate application from screen management functions, and support customization features for the end user.

Analysis and Design Engines

A new generation of analysis and design tools, called analysis and design engines, uses a rule-based architecture that enables the tool to be customised for any analysis and design method. Using these advanced CASE tools an analysis and design method such as the SADT can be supported by building the appropriate graphical notation entering the rules that supports analysis and design using the method/ in essence analysis and design engines enable a software engineer to customise the tool to meet the need of specific (and possible obscure) method/ all the tools in this category support SA/SD\ but they can also support SD, DSSD, SADT, HOOD, and a variety of other methodologies.

PROGRAMMING TOOLS

The programming tools category encompasses compilers, editors, and debuggers that are available to support moat conventional programming languages. In addition, object-oriented (0-0) programming environment fourth-generation languages, application generators, and database query languages also reside within this category.

Conventional Coding Tools

There was a time when the only tools available to a software engineer were conventional coding tools-compilers, editors and debuggers. Pressman and Herron [PRE91] discuss this when they state:

There's an old saying: When the only tool that you have is a hammer, every problem looks like a n. "think about it. You can use a hammer to pound nails, but if its the only tool that you have, you can also use it to pound screws (sloppy, but workable), bend metal (noisy, but workable, punch a hole in wood or concrete very sloppy but possible)... We do the best we can by adapting the tools that we have on hand.

For almost 30 years, the only tools available to programmers were conventional coding tools and, therefore, every software engineering problem looked like n coding problem. Today, conventional tools continue to exist at the front lines of software development, but they are supported by all the other CASE tools discussed in this chapter.

Fourth-Generation Coding Tools

The thrust toward the representation of software applications at a higher level of abstraction has caused many developers to move headlong toward: fourth-generation coding tools. Database query systems, code generator and fourth-generation languages have changed the way in which systems are developed. There is little doubt that the end goal of CASE is automatic code generation that is, the representation of systems at a higher level of abstraction than conventional programming languages. Ideally such code generation tools will not only translate a system description into an operational program but also serve to help verify the correctness of the system specification so that the resulting output will conform to a user requirement.

Fourth-generation languages are already used widely in information systems applications. It is not unusual to read claims such as: "Rank Xerox in the U.K... created an application with 350.000 lines of COBOL code... yet it was created with three full-time people and one part-time person in ten weeks" [CAS89]. Although such accomplishment are possible in very limited domains of applicability they represent a harbinger of things to come in broads application areas. We are already beginning to see the first code generation tools appear in the engineered products and systems market (most focus on Ada). As the 1990s progress, it is likely that less and less source code will be "written" manually.

Although fourth-generation languages. code generators, and application generators (e.g., database query systems) all enable a software engineer to specify a system at a high level of abstraction, each of these tools differ m important ways [F0R87]. Referring to Figure 5, a fourth-generation language is input directly to a 4GL interpreter. The interpreter translates the 4GL into executable code. The input to a code generator is a procedures specifications language (PSL) (a metalanguage). The procedural specification language is then processed by one or a number of code generation modules that translates the PSL into the appropriate programming language. An application generator uses a central database or data dictionary interactive menu-driven features, and application-specific rules to create software that addresses a narrow application domain.

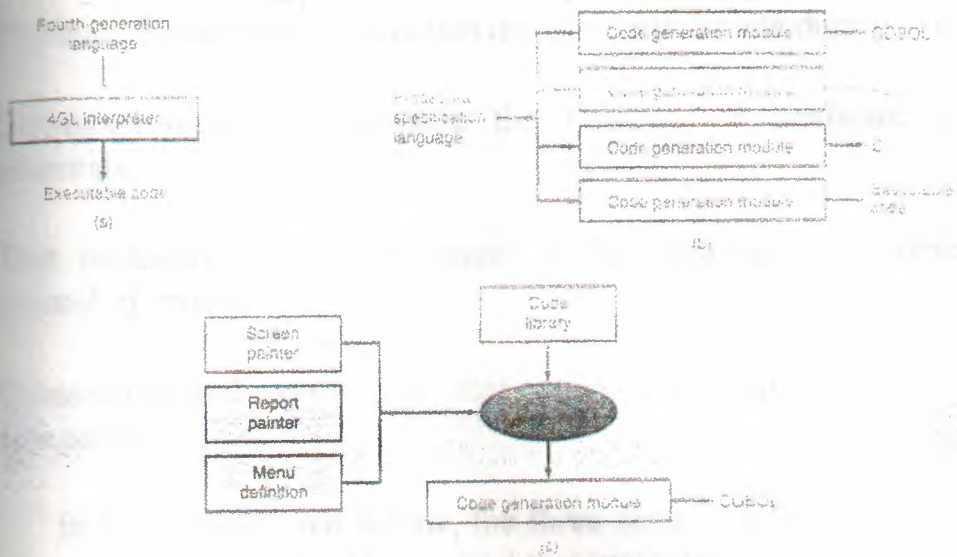


Figure 5. Fourth generation tools.

Object-Oriented Programming Tools

Object-Oriented programming is one of the "hottest" technologies in software engineering. For this reason, CASE vendors are rushing new tools for object oriented software development to the market.

Object-oriented programming environments are tied to a specific programming language (e.g. C++, Eiffel, Objective-C, or Smalltalk). A typical 0-environment incorporates third-generation interface features (mouse, windows, pull-down menus, context-sensitive operations, multitasking) with specialised functions such as the "browser"- a function that enables the software engineer to examine all objects contained in an objects contained in an object library to determine whether any can be reused in the current application.

INTEGRATION AND TESTING TOOLS

In its directory of software testing tools. Software Quality Engineering [SQE90] defines the following testing tools categories:

- Data acquisition-tools that acquire data to be used during testing
- Static measurement-tools that analyse source code without executing test cases.

- Dynamic measurement-tools that analyse source code during execution
- Simulation-tools that simulate the function of hardware or other externals.
- Test management-tools that assist in the planning, development, and control of testing.
- Cross-functional tools-tools that cross the bounds of the above categories.

In the sections that follow, the three most widely used testing tools categories are discussed. It should be noted that many testing tools have features that span two or more of the above categories.

Static Analysis Tools

Static testing tools assist the software engineer in deriving test cases. Three different types of static testing tools are used in the industry: code-based testing tools, specialised testing languages, and requirements-based testing tools.

Code-based testing tools accept source code (or PDL) as input and perform a number of analyses that result in the generation of test cases. Using a description of the program input and procedural design as a guide, static testing tools derive test cases using path coverage, condition testing, and data flow criteria.

Specialised testing languages (e.g. ATLAS) enable a software engineer to write detailed test specifications that describe each test case and the logistics for its execution. However, such tools do not assist the tester in designing the test cases.

Requirements-based testing tools isolate specific user requirements and suggest test cases (or classes of tests) that will exercise the requirements. To work properly, tools in this subcategory must have access to a formal specification for the software.

In most cases, static testing tools will document and catalogue tests (e.g. tests to exercise a particular type of input). They will conduct comparisons of test output to note differences between expected and actual results.

Dynamic Analysis Tools

Dynamic testing tools interact with an executing program, checking path coverage, testing usurious about the value of specific variables, and other wise instrumenting the execution flow of the program. Dynamic tools can be either intrusive or nonintrusive . an intrusive tool changes the software to be tested by inserting probes (extra instructions) that perform the activities mentioned above. Nonintrusive testing toots use a separate hardware processor that runs in parallel with the processor containing the program that is being tested.

Most tools in the dynamic analysis category produce reports that indicate the number of times blocks of statements have been executed (path coverage analysis) and the average execution time for blocks of statements (performance analysis).

Another type of dynamic testing tool is sometimes called a capture/playback tool [POS89]. In capture mode, a capture/playback tool records all information flow at a particular point in a program's execution cycle. Often, the point of capture occurs immediately after interactive input is provided, i.e., the capture point "sits right behind the screen." Later, when the tool is placed in play6ack mode, the program can 6e restarted at the point of capture and will execute as if the original data were input to the program. Capture/playback tools are quite useful for creating regression test suites for highly interactive programs.

A dynamic testing tool can be used in conjunction with a static testing tool. The static tester is used to derive the test cases that are then monitored by the dynamic tool.

Test Management Tools

Test management tools. are used to coat control-coordinate software testing for each of the major testing steps. Tools in this category manage and coordinate regression testing, perform camparisons that ascertain differences between actual and expected output, and conduct batch testing of programs with interactive human-computer interfaces.

In addition to the functions noted above. many test many test management tools also serve as generic test drivers. A test driver reads one or more test cases from a testing file, formats the test data to conform to the needs of the software under test, and then invokes the software to be tested. Testing tools in this subcategory are customised by the tester to meet specialised testing needs.

Finally, test managers sometimes work in conjunction with requirements tracing tools (Section 5.2) to provide requirements coverage analysis for testing. Reading each teat case in sequence, the requirements coverage analyser attempts to determine (based on information that describe the purpose of the test case) which software requirements are addressed the test. A cross-reference matrix is often used to indicate which tests address what requirements.

PROTOTYPING TOOLS

Prototyping is a widely used software engineering paradigm, and as such, any tool that supports it can legitimately be called a prototyping tools. For this reason many of the CASE tools discussed in this chapter can also be included in this category.

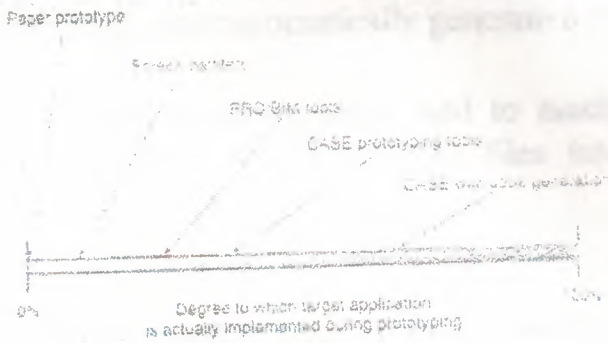


Figure 6. Prototyping tools.

All prototyping tools reside somewhere on the implementation spectrum illustrated in Figure 6. At the low end of the spectrum, tools exist for the creation of a "paper prototype". A PC or workstation-based drawing tool can create realistic screen images that can be used to illustrate system function and behaviour to the customer. These images can not be executed. Screen painters enable a software engineer to define screen layout rapidly for interactive applications. In some cases a screen painter will also generate the source code to create the screen. More sophisticated CASE prototyping tools enable the creation a data design couple with both screen and report layout. Many analysis and design tools have extensions that provide a prototyping option. PRO/SIM tools have prototyping features.

As prototyping tools evolve, it is likely that some will become domain-specific. That is, the tool will be designed to address a relatively narrow application area. Prototyping tools for telecommunications, aerospace applications, factory automation, and many other areas may become common-

place by the mid-1990s. such tools will use a knowledge base that "understands" the application domain, facilitating the creation of prototype systems.

MAINTENANCE TOOLS

CASE tools for software maintenance address an activity that currently absorbs approximately 70 percent of all software related effort. The maintenance tools category can be subdivided into the following functions:

- Reverse engineering to specification tools-take source code as input and generate graphical structured analysis and design models, where-used lists, and other design information.

- Code restructuring and analysis tools-analyse program syntax, generate a control flow graph, and automatically generate a structured program.

- On-line system re-engineering tools-used to modify on-line database systems (e.g., convert IDMS or DB2 files into entity-relationship format)

The above tools are limited to specific programming languages (although most major languages are addressed) and require some degree of interaction with the software engineer.

Next generation reverse engineering and re-engineering tools will make much stronger use of artificial intelligence techniques, applying a knowledge base that is application domain-specific (i.e., a set of decomposition rules that will apply to all program in a particular application area such as manufacturing control or aircraft avionics). The AI component will assist in system decomposition and reconstruction, but will still require interaction with s software engineer throughout the re-engineering cycle.

Reverse Engineering Tools

Reverse engineering tools perform a post-development analysis on an existing program. Like testing tools, reverse engineering tools can be categorised as static or dynamic.

A static reverse engineering tool (by far, the most common) usages program source code as input and analyses and extracts program slicing. The

ware engineer specifies the type of program structure (data declaration, loops, ther logic) that are of interest and the reverse engineering tool removes traneous code, enabling only code of interest to be represented. Dependency alysis tools perform most of the functions already discussed, but it addition, ols in this subcategory build graphical dependency maps that show the link tween data structures, program components, and other user-specifies program aracteristics. Static reverse engineering tools have been called "code sualisation" tools [OMA90]. In fact. by enabling the software engineer to "isualise" the program, such tools greatly improve the quality of changes that are ade and the productivity of the people making them.

Dynamic reverse engineering tools monitor the software execute and uses formation obtained during monitoring to build a behavioural mode of the ogram. Although such tools are relatively rare the provide important formation for software engineers who must maintain real-time software or mbedded system.

Re-engineering Tools

Although re-engineering tools offer significant promise, relatively few ndustry quality tools are in use today. Existing re-engineering tools can be ivided into two subcategories--code restructuring tools and data re-engineering ools. Code restructuring tools accept unstructured source code as input, perform he reverse engineering analysis described in Section 22.11.1, and then estructure the code to conform to modern structured programming concepts. Although such tools can be useful, they focus solely on the procedural design of a program.

Data re-engineering tools work at the other end of the design spectrum. Such tools assess data definitions or a database described in a programming language (usually COBOL) or database description language. They then translate he data description into graphical notation that can be analysed by a software engineer. Working intellectively with the re-engineering tool, the software engineer can modify the logical structure of the database, normalise the resultant files, and then automatically regenerate a new database physical design. The ools may use an expert system and knowledge base to optimise the re-engineered software for improved performance.

FRAMEWORK TOOLS

The industry trend toward I-CASE environments will continue to gain momentum during the 1990s. framework-tools software tools that provide database management, configuration management and CASE tools integration capabilities-are the first thrust in the IPSE direction.

Tools in this category exhibit functional components that support data interface, and tools interaction. Most implement an object-oriented database with an internal tool set for establishing smooth interface with tools from other CASE vendors. Most framework tools provide some configuration management capabilities, enabling the user of the tool to control changes to all the configuration items created by all the CASE tools that are integrated with the framework tool. The key components of framework tools will be discussed later.

CASE AND AI

A few CASE tools have limited expert system capabilities, but the vast majority of existing CASE tools make little use of artificial intelligence techniques. Most toots that do make limited use of AI employ the technology to check the graphical correctness of analysis and design models applying design rules inherent to a particular analysis and design method to the models that have been created by the software engineer. However, the real promise of CASE-AI lies elsewhere.

Researchers are evaluating programming environments that make use of analysis design agents intelligent tools that aid in the analysis, design, and testing of computer-based systems. Rather than simply evaluating the model that a human has created. an agent will assist the software engineer in his or her problem solving activities. Such agents must be domain-specific, accessing a knowledge base about the characteristics of a limited class of applications and being capable of using this knowledge base to guide the software engineer in analysis, design, or testing. The problem, of course, is in the definition of a knowledge base for software engineering. Although we are still a number of years away from such "agents," the future for CASE-AI is promising .

CONCLUSION

Computer-aided software engineering tools span every step in the software engineering process and those umbrella activities that are applied throughout the process. CASE comprises a set of building blocks that begin at the hardware and operating system software level and end with individual tools.

In this chapter we have considered a taxonomy of CASE tools. Categories encompass both management and technical activities and span most software application areas. Each category of tool has been considered as a point solution. In the next chapter, we consider ways in which individual tools are integrated to form an environment.

As the years pass, CASE will became part of the fabric of software engineering. Just as mechanical and electrical engineers rely on CAD/CAE/CIM for the analysis and design of high-technology products, software engineers will rely on CASE for the analysis, design and testing of computer-based systems for the twenty-first century.

INTEGRATED CASE ENVIRONMENTS

Computer-aided software engineering (CASE) is changing the industries approach to software development. Although benefits can be derived from individual tools that address separate software engineering activities, the real power of CASE can only be achieved through integration. Gene Forte [FOR89a] makes this point clear when he states:

Tool integration is among the most often discussed and debated topics in software engineering. Justly so, since no other technical or strategic issue is likely to have as much impact on the evolution of [software] technology and the CASE industry…

While individual CASE tools each contribute…, the promise of CASE really lies in the potential to integrate many tools into an integrated environment.

The benefits of integrated CASE (I-CASE) include (1) the smooth transfer of information (models, programs, documents, data) from one tool to another and one software engineering step to the next; (2) a reduction in the effort required to perform umbrella activities such as software configuration management, quality assurance, and document production; (3) an increase in project control that is

- Allow direct, non-sequential access to any tool contained in the environment.

- Establish automated support for a procedural context for software engineering work that integrated the tools and data into a standard work breakdown structure.
- Enable the users of each tool to experience a consistent look and feel at the human-computer interface.

- Support communication among software engineers.

- Collect both management and technical metrics that can be used to improve the process and the product.

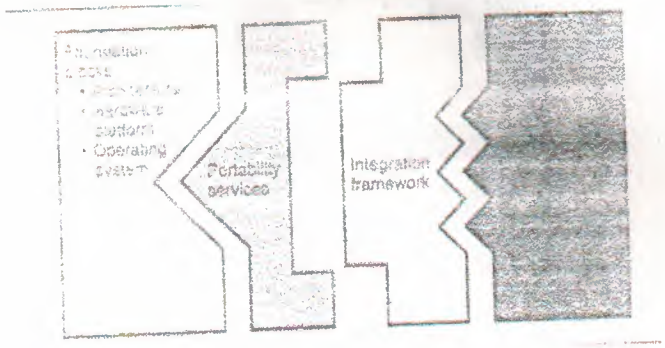


Figure 7. I-CASE

To achieve these requirements, each, of the building blocks of a CASE architecture must fit together in a seamless fashion. Referring to Figure 7, the foundation building blocks-environment architecture, hardware platform, and operating system "must be joined" through a set of portability service to an integration framework that achieves the requirement notes above. For the remainder of this chapter. We examine the integration framework in greater detail.

INTEGRATION OPTIONS

CASE tools can be integrated in many different ways. At one end of an integration spectrum, a CASE tool is used in complete isolation. A limited number of software configuration items (documents, program, or data) are created and manipulated by a single tool and output is in the form of hardcopy text and/or graphical documentation. In a sense, linkage, to the rest of the software development environment is by paper via the developer.

In reality, few CASE tools are used in total isolation. The following integration options (Figure 8) are available:

Data Exchange Most tools have at least the ability to export information they gather and create in the form of an instructed file with a published format: This enables a point-to-point data exchange (Figure 8) between one CASE tool and some other tool, usually with a transmitting "filter" interposed. This preserves the information contained in the tool, eliminating the need to re-enter existing elements of the specification of design and preventing typographic errors from being introduced unnecessarily.

Many translators have been developed through the manual cooperation of the tool vendors involved end are available directly from them. In addition, many

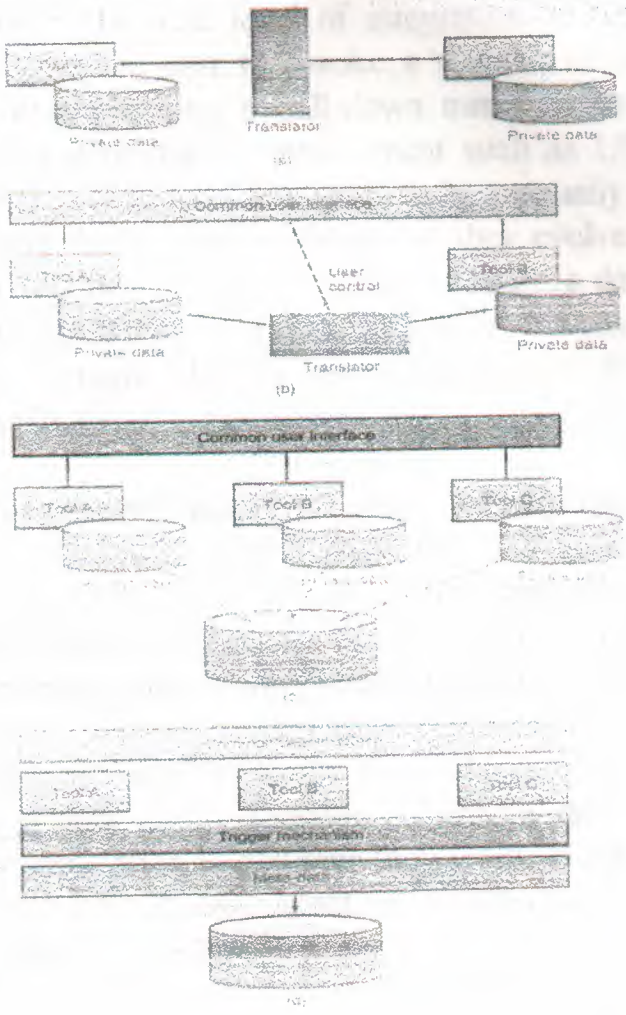


Figure 8. Levels of CASE integration.

translators have been developed by consultants and users and are available for purchase or through "shareware" exchanges.

The drawback of the point-to-point data exchange is that usual only portion of the data exported can be used by the receiving tools since designed to be fully compatible. In addition, as the software evolves, it can become time consuming to transfer files each time a small exchange is made. Versions can easily get "out of sync."

When many tools are used on a project, the number of point-to-point transfers can become unacceptable large. Finally, the transfer is normally in one direction only. There is no potential for reflection changes in to directions, and it is difficult to make cross-document checks and maintain integrity (configuration control) of the configuration across the various tools that are used.

Common Tool Access The next level of integration ie common tool access (Figure 8), which allows the user to invoke a number of different tools in a similar manner, for example form a pull-down menu in the operating system window manager. In a multitasking environment such as UNIX or OS/2. this means a user can open several tools simultaneously, manually coordinating input to them and comparing design representations as they evolve. For example, the user might display a data flow diagram, a structure chart, a data dictionary and a source code segment maintained by different tools. In this environment, the tool-to-tool data exchange might also be simplified by invoking the translation procedure with a simple menu or macro selection.

Common Data Management Data from a variety of tools can be maintained in a single logic database (Figure 8), which phsycally may be either centralised or distributed. This simplifies the exchange of information and improves the integrity of the shared data, since each tool always has immediate access to the latest software engineering information. Access rights in a team environment can also be controlled and version management facilities may also be available, although these will be activated manually via a check-in, check-out procedure. Typically, there is a data merge function to enable developers working on different parts of an application to combine they work. If the tool set hes cross-project checking capability, it can detect inconsistencies among the different developed contributions.

Although the data from multiple tools are managed together at the common data management level, the tools have no explicit understanding of each other's internal data structures and design representation semantics. Consequently a discrete translation step (usually invoke manually) is steel required to enable one tool to use the output from another tool.

Data Sharing Tools at the data sharing level have compatible data structures and semantics and can directly use each other's data without translation. Each tool is designed to be compatible with all other tools in the data sharing environment. For this reason most data sharing occurs tool among from a single vendor or where strategic relationships between vendors have been formed to produce an integrated tool at set, sometimes at the request of large customers. Unfortunately, there are few official standards in place to provide the common ground for industry-wide data sharing among CASE tools. However, de facto standards, suggested by large CASE vendors (e.g. DEC, Hewlett-Packard, IBM, or Sun), may provide the industry with some degree of coordination for data sharing.

Interpretability Tools that combine the characteristics of common access and data sharing are interpretable. This represents the highest level of integration among individual tools. However there are other properties of the overall CASE environment that can be added to improve the effectiveness of the software development process.

Full Integration To achieve complete integration (Figure 8) of the CASE environment, two additional facilities are needed: meta-data management and a control facility. Metadata is information about software engineering data produced by the individual CASE tools. Metadata includes.

- Object definition (types, attributes, representations, and valid relationships)
- Relationship and dependence among of arbitrarygranularity (i.e., a process on a DFD diagram, a single entity, or a subroutine code fragment)
- Software design rules (e.g.. the correct ways to draw and balance a data flow diagram)
- Work flow (process) procedures (standard phases, milestone deliverables, etc.) and events (reviews, completions, problem reports, change requests, etc.)

Often the rules and procedures portion of the metadata is defined in the form of a rule base to facilitate its modification as the software development process evolves. For example, a new design method might alter design rules for representation and change work flow process standards.

The control facility enables the individual tools in notify the rest of the, environment (other tools, the metadata manager, the data manager, etc.) of significant events and to send requests for action to other tools and services via a trigger. For example a design tool might notify configuration management tool that a new version of a design document has been created and cause the configuration management tool to do a cross-document consistency check. The control facility helps to maintain the integrity of the environment and also provides a means to autromate standard process and procedures

The trigger facility might be embedded within a closed repository environment, or it might be visible to individual tools through a programmatic interface and message-passing mechanism.

THE INTEGRATION ARCHITECTURE

Using CASE tools, corresponding methods, and a procedural framework defined by the software engineering paradigm that has been selected a pool of software engineering information is crested. The integration framework facilitates transfer of information into and out of the pool. To accomplish this, the following architectural components must exist: A database must be created (to store the information): an object management system must be built (to manage changes to the information); a tools control mechanism must be constructed (to coordinate the use of CASE tools); a user interface must he available to provide a consistent pathway between actions made by the user and the tools contained in the environment. Most models (e.g., [WAS89], [FOR9O] of the integration framework represent these components as layers. A simple model of the framework, depicting only the components noted above, is shown in Figure 9.

The user interface layer (Figure 9) incorporates a standardised interface tool kit with a common presentation protocol. The interface tool kit contains software for human-computer interface management and a library of display object. Both provide consistent mechanism for communication between the interface and individual CASE tools. The most commonly used tool kit for CASE is the X-Window System [MIK90]. The representation protocol is the set of guidelines that gives all CASE tools the same look and feel. Screen layout conventions, menu names and organisation, icons, object names, the user of the keyboard and mouse, and the mechanism for the tools access are all defined as part of the representation protocol.

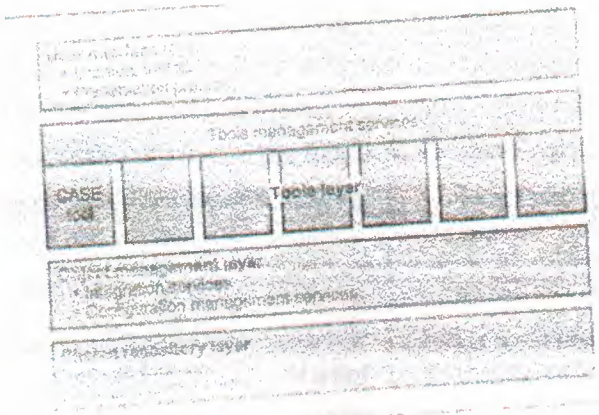


Figure 9. Architectural model for the integration framework.

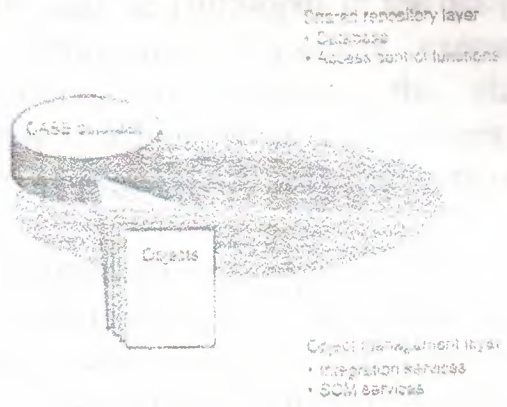


Figure 10. The layers that achived data integration.

The tools layer incorporates a set of tools management services with the CASE tools themselves. Tools management services (TMS) control the behaviour of tools within the environment. If multitasking is used during the execution of one or more tools, TMS perform multitask synchronisation and communication co-ordinates the follow of information from the repository and object management system into the tools, accomplishes security and auditing functions, and collects metrics on tool usage.

The object management layer (OML) performs the configuration management functions described before. In essence, the software in this layer of the framework architecture provides the mechanism for the tools integration. Every CASE tools is "plugged into" the object management layer. Working in

conjunction with the CASE repository the OML provides integration services a set of standard modules that couple tools with the repository. In addition, the OML provides configuration management services by enabling the identification of all configuration objects, performing version control, and providing support for change control, audits, and status accounting.

The shared repository layer is the CASE database and the access control functions that enable to object management layer to interact with the database. Data integration is achieved by the object management and shared repository layers (Figure 10) and is discussed in greater detail later in this chapter.

TOOLS INTEGRATION

when an integrated CASE environment is considered the mechanism for the integration of CASE tools will be implemented differently depending on the architecture, the platform, and the philosophy of the designer of the environment. However, all CASE environment implement execution. mechanisms and communication mechanisms. To illustrate the characteristics of these mechanisms, the Portable Common Tools Environment (PCT) model-one of a number of standards for integrated CASE environments will be used.

Within PCTE, execution and communication mechanisms are referred to as basic mechanisms-functions that are defined to manipulate "entity" that exist in the software development context. Entities include both objects (e.g., data, source code, documents, devices) and the tools that operate on the objects.

Most I-CASE environments am designed to accommodate a multitasking operating system in which s number of different tools can be executing at the same time. For example, a compilation of one module can be invoked at the same time that modifications are being made to the design of another module. In addition, the completion of a task performed by one tool might lead automatically to the execution of another tool, if appropriate process activation mechanisms (triggers) are present.

Execution mechanism provide "a uniform way ta start a process from its static context regardless of whether it is an executable or interpretable program" [THO89]. In addition, these mechanisms provide features for suspending, resuming, and terminating a process. In this context a "process" can be viewed as a CASE tool.

Communication mechanisms manage interprocess communication by establishing message queues that enable different tools the communicate with one another. For example the completion of a task performs by CASE tool A may result in an "event" that leads to the initiation of CASE tool B. to invoke B, an execution mechanism must be used, but to pass information from tool A to tool B requires a communication mechanism.

It should be noted that basic mechanisms use sophisticated communication and coordination capabilities that are often associated with operating system functions. The challenge for I-CASE environment developers is to implement these mechanisms in a way that decouples the environment from a specific operating system or, as a minimum provides a layer between the environment itself and operating system internal.

Because the environment architecture for CASE is a distributed network, the mechanism described in this section must be capable of being implemented in a networked environment. A distribution mechanism enables the basic mechanism to be distributed across a network and also provides the following capabilities [THO89]: (1) network administration and supervisor off all workstations connected to the network; (2) management of each network node (i.e., each workstation can be represented as an object in the repository and has an attribute the workstation directory; (3) "transparent distribution" of execution and communication functions.

DATA-TOOL AND DATA-DATA INTEGRATION

Data integration can be examined from two different points of view: (1) Data-tool integration considers the level of integration between CASE tools and the data that are producted by the tools (and people) throughout the software engineering process; (2) data-data integration examines the level of integration among the information items themselves.

In order to achieve the types of integration described above, it is necessary to define an abstraction that enables us to connect information entities (configuration objects) and provide some mechanism for establishing the relationships between these entities. This lead to an "object-oriented view" of the CASE database. Elements of the software configuration (e.g., programs, documents. and data) are treated an objects to be manipulated as part of the software engineering process.

Once an object-oriented abstraction is established, the tool-data integration ew defines agents (e.g., users or tools) that operate on objects. (programs, cuments, data). It is important to note that the operation: implied by this scussion span a broad range of functionality. Operations can be as simple as s xt editing process or as complex as a sophisticated software engineering ethod. For example, by combining the appropriate design information, it is latively easy to create an object that we might characterised, as a design cument. The software engineer (e.g., the user of the CASE systems) can erate on design document in a number of different ways. For instance, the ftware engineer can review design document in a purely manual fashion. Yet e importance of the review operation can not be over-emphasised. The ftware engineer can also edit design document analyse the certain aspects of chitectural and/or procedural design, refine or elaborate some aspects of the sign, and transmit the design to another CASE tool that might generate code or ovide traceability back to requirements. Each of the italicised terms in the eceding discussion represents an operation that can be applied to an object. ata-tool integration is accomplished as each of these operations is implemented the a CASE environment. It is fear to say that not all operations will be plemented as to functions. In fact some of the most important operations may e manual but steel apply to the overall concept of integration.

Architecturally, data-tool integration was discussed in Section 23.2 and nges from simple tool-to-tool data exchange to a complete I-CASE environ- ent. As data-tool integration becomes more sophisticated, the complexity of the ared repository and the number of layers in the environment model must both crease.

Data-data integration can be modelled using entity-relationship techniques. A configuration object (entity) is always related to one or more other nfiguration objects. For example, source code is related to the design chitectural model and to one or more user requirement. The source code is also lated to one or more test cases. The relationships described above as well as a w others, can be depicted as an E-R model shown in Figure 11 referring to the gure, the relationships show in the diamonds are implemented using one or ore against (CASE tools, people) and their corresponding operations.

The relationships depicted in the E-R diagram can also be used to represent different software versions. Recalling discussion of version control in previous, the relationships defined to achieved data-data integration propagate across the evolution graph for the software. Each version maintain the some generic relationships among configuration objects.

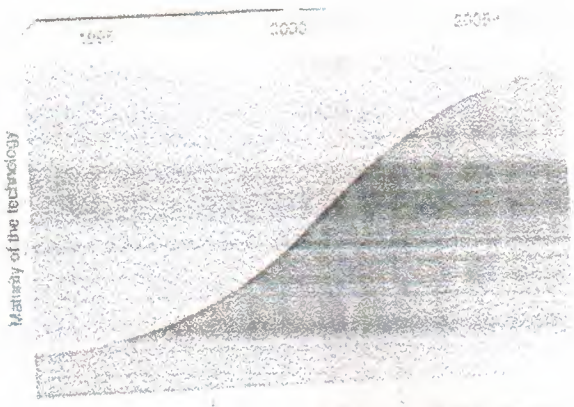


Figure 12. The 5-5-5 rule.

THE CASE REPOSITORY

Webster's dictionary [WEB74] defines the word repository as "any thing or person though of as a center of accumulation or storage." During the early history of software development, the repository was indeed a person-the programmer who had to remember the location of all the information to a software project; who had to recall information that was never written down and reconstruct information that had been lost.. Sadly using person as "the center for accumulation and storage" (also it confirmation Webster's definition) does not work very well. Today the repository is a "thing"-a database that acts as the center for both accumulation and storage of software engineering information. The role of the person (the software engineer) is to interact with the repository using CASE tools that are integrated with it.

In this project, a number of different terms are used to refer to the storage place for software engineering information: CASE database project database, integrated project support environment (IPSE) database, and prository. Although there are subtle differences between some of these terms, all refer to the thing that is through of as the center for accumulation and storage.

The Role of the Repository in 1-CASE

The repository for an I-CASE environment is the set of mechanisms and data structure that achieve data-tool and data-data integration. It provides the

functions of a database management systems, but in addition, the repository performs or precipitates the following functions [FOR89b].

- Data Integrity. Includes a function to validate entries to the repository ensures consistency among related objects, and automatically performs "cascading" modifications when a change to one object demands changes to objects that are related to it.

- Information Sharing. Provides a mechanism for sharing information among multiple developers and between multiple tools, manages and control multi-user access to data, and locks/unlock objects so that changes are not inadvertently overlaid on one another.

- Data-Tool Integration. Establishing a data model that can be accessed by all tools in the I-CASE environment, controls access to the data, and performs appropriate configuration management functions.

- Methodology Enforcement. Defines a specific paradigm for software engineering that is implied by the E-R model of data stored in the repository; as a minimum, the relationships and the objects define set of steps that must be conducted to build the content of the repository.

- Document Standardisation. Leads directly to a standard approach for the creation of software engineering documents by creating definitions for objects in the database.

To achieve these functions, the repository is defined in terms of a meta-model. The meta-model determines how information is stored in the repository, how data can be accessed by tools and viewed by software engineers how well data security and integrity can he maintained; and how easily the existing model can be extended to accommodate new needs [WEL89].

The meta-model is the template into which software engineering information is placed. Earlier in this chapter we discussed the entity-relationship - attribute meta-model, but other more sophisticated models are also under consideration. A detailed discussion of these models is beyond the scope of this project. For further information, the interested reader should see Welke [WEL89].

Features and Content

The features and content of the CASE repository are best understood by looking at the repository from two Perspectives: what is to be stored in it and what specific services it provides. In general, the types of things to be stored in the repository include the following:

- The problem to be solve
- Information about the problem domain
- The system solution as it emerges
- Rules and instructions pertaining to the software process (methodology) being followed
- The project plan, resources, and history
- Information about the organisational context

A detailed list of types of representations documents, and deliverable's that are stored in the CASE repository is included in Table 1.

A robust CASE repository provides two different classes of services: (1) the same types of services that might be expected from any sophisticated database management system, and (2) services that are specific to the CASE environment.

Many repository requirements are the same as those of typical applications built on a commercial database management system. In fact most of today's CASE repositories employ a DBMS (usually relational or object-oriented) as the basic data management technology. The standard DBMS feature of CASE repository supporting the management of the software development information include:

Non-redundant Data Storage. The CASE repository provides a single place for the storage of a11 information pertinent to the development of software systems, eliminating wasteful and potentially error-prone duplication

High-Level Access. The repository provides a common data access mechanism so data handling facilities do not have to be duplicated in each CASE tool.

Data Independence. CASE tools and the target applications are isolated from physical storage so they are not affected when the configuration is changed.

Transaction Control. The repository manages multipart interaction manner that maintains the integrity of the data when there are concurrent. users and in the event of a system failure. This usually implies record locking, two-stage commits, transaction logging, and recover procedures.

TABLE 1

CASE REPOSITORY CONTENTS [FOR89b]

Enterprise information	[illegible]
Organizational structure	[illegible]
Business area analyses	Test cases
Business functions	[illegible]
Business rules	[illegible]
Process models (scenarios)	[illegible]
Information architecture	[illegible]
Application design	
Methodology rules	
Graphical representations	[illegible]
System diagrams	Project plans
Naming standards	Work breakdown structure
Referential integrity rules	Estimates
Data structures	Schedules
Process definitions	[illegible]
Class definitions	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	
[illegible]	
[illegible]	
Binary images	
Configuration dependencies	
Change information	

Security. The repository provides mechanisms to control who can view and modify the information contained within it. At a minimum the repository should enforce multilevel passwords and permission level assigned by individual users. The repository should also provide assistance for automatic backup and restore,

and archiving of selected group of information, for example, by project or application.

Ad Hoc Data Queries And Reports. The repository allows direct access to in contents through a convenient user interface such as SQL or a form oriented "browser" enabling user-defined analysis beyond the standard reports provided with the CASE tool set.

Openness. Repositories usually provide a simple import/export mechanism to enable bulk loading or transfer. The interfaces are usually ASCII file transfer or a standard SQL interface. Some repositories have high-level interfaces that reflect the structures of the meta-models.

Multiuser Support. A robust repository must permit multiple developed to work on an application at the same time. It must manage concurred access to the database by multiple tools and users with access arbitrary and locking at the file or record level. For environments based networking, multiuser support also implied that the repository can interface with common networking protocols and facilities.

The CASE environment also makes special demand on the repository that go beyond what is directly available in a commercial DBMS. The special features of CASE repositories include:

Storage Of Sophisticated Data Structures. The repository must access modate complex data types such as diagrams, documents and files well as simple data elements. A repository also includes an meta-model describing the structure relationships the data stored in it. The metamodel must be extensible that new representations and unique organisational the repository not only stores models and description the systems under development but also associated metadata (i.e., additional information describing the software engineering data types then a particular component was created, what other components it depends upon).

Integrity Enforcement. The repository information model also contained describing valid business, rules and other requirements on information being entered into the repository (directly a CASE tool). A trigger may be employed to activate the rules associated with an object whenever it is modified, making it possible to check the validity of design models in real time.

Semantic-Rich Tool Interface. The repository information model (model) contains semantics that enable a variety of tools ta interpret that meaning of the data stored in the repository. For example, a data diagram created by a CASE

is stored into the repository in a form used on the information model and independent of any internal representations used the tool itself. Another CASE can then interpret the contents of the repository.

The following repository to leave the normal development environment. ignment and can also be by electrons are Problem reports, maintenance tasks, ange authorisation, and repair status can be coordinated and monitored via ols accessing the repository.

rsioning As a project processes, many versions of individual work products ll be created. The repository must be able to save all of these versions to able the effective management of product releases and to permit developers to back to previous versions during testing and debugging. Versioning is done h compression algorithm to minimise storage allocation, and permits the eneration of any previous version with some processing overhead.

The CASE repository must be able to control a wide variety of object pes including text, graphics, bit maps, complex documents and unique objects e screen and report definition, object files and test data and results. A mature ository tracks versions of object with arbitrary levels of granularity; for mple, a single data definition or a cluster of modules can each be tracked.

To support parallel development, the version control mechanism should mit multiple derivatives (variants) from a single predecessor. Thus a developer uld be working on two possible solution to a design problem at the same time, th generated from the same staring point.

Dependency Tracking and Change Management. The repository man These clude relationships between enterprise entities and processes, among the parts application design between design components and the enterprise information chitecture. between design elements and deliverable's, and so on. Some of ese relationships are merely associations, and some are dependencies ndatory relationships. Maintaining these relationships among development jects is called link management.

Link management helps the repository mechanism ensure that design rmation is correct by keeping the various portions of a design synchronised. r example, if a data flow diagram is modified, the repository can detect ether relates data dictionaries, screen definitions, and code modules also quire modification and can bring affected components to the developer's ention.

While CASE environments initially implemented link management at the file level, the trend is toward management at the object level where an object can be any level of aggregation from a single data element to an application comprising many files. Such a robust implementation requires a generalised object management facility independent of the underlying file

Requirements Tracing A special function depending on link management is requirements Tracing. This is the ability to track all the design components and deliverable's that result from a specific requirement specification (forward tracking), as well as the ability to identify which requirement generated any given deliverable (backward tracking).

Configuration Management Another function depending on link management is configuration management. A configuration management facility works closely with the link management and versioning facilities to keep track of a series of configurations representing specific project milestones or production releases. Version management provides the needed versions, and link management keeps track of interdependencies. For example, configuration management often provides a build facility to automate the process of transforming design components into executable deliverable's. Active link management associated with CASE design tools through the repository can ensure that all items affected by changed design representations will be properly regenerated as needed without explicit instructions from the developer. For example, a change to an E-R diagram may affect a screen definition and a database schema. Both might be automatically updated or called to the developer's attention. The configuration management facility might then initiate regeneration of the change and re-links the changed modules. While configuration target code to reflect the automate and re-link the changed modules.

Audit Trails Related to change managements the need for an audit trail that establishes additional information about when, why, and by whom changes are made. Actually this is not a difficult requirement for a repository that has a robust information model. Information about the source of changes can be entered as attributes of specific object in.

Repository Standards

A number of different standards efforts are underway for I-CASE environments and the CASE repository. In some cases, a proposed standard goes beyond the definition of a repository to consider many different aspects of an integrated environment. In the United States, a number of proposed standards are competing

dominance. In Europe, a single standard has been adopted. Similarly Japan and other far-east countries have a adopted a single (but different) standard for I-CASE. Each of the standards efforts is described briefly here so that the reader can gain a basic understanding of work that is underway.

Information Resource Dictionary Standard (IRDS), ANSI (X3.13 1988). The only formally approved ANSI standard presented in the section, IRDS was originally developed as a standard definition for requirements dictionaries (this standard can also be used for repositories). It focuses on the management of corporate information resources and is characterised using a multilevel meta-model. This standard help in the creation of "bridges" between complementary tools such analysis/design tools and code generators and in CASE tool portability across different platforms.

Alberton/DEC Tool Integration Standard (ATIS). Originally developed by Alberton Technology (a developer of framework tools) and Digit Equipment Corporation, ATIS has been adopted by the CASE Integration Standards Committee of the Software Productivity Consortium ATIS focuses on the definition of a repository architecture and address SCM, tool integration, data security, and portability across platforms.

Common Ada Interface Standard (CAIS). Focusing primarily on tools Ada software development, the CAIS standard defines interface twenty tools that will comprise the Ada development environment.

Portable Common Tools Environment (PCTE). PCTE was developed use by the European software development community (the ESPR project) and has been adopted by the European Computer Manufactures Association (ECMA). PCTE is an interface standard and architecture model for CASE. It address probability, contingency control, network distribution, data architecture, and the user interface in the context of the CASE environment [THO89].

Software Industrialised Generator and Maintenance Aids (SIGMA). SIGMA is similar to PCTE in intent and scope and has been adopted Japan and other far-East countries.

Electronic Design Interchange Format (EDIF). This standard focuses data formats for information exchange between CASE tools (and other programs that want to exchange data). A CASE tool that product output information in this format can easily transmit the information other tools that accommodate input in EDIF format.

In addition to the standards introduced above, every major computer vendor has proposed "solutions" for I-CASE. A list of some of the common frameworks follows:

- DSEE-Apollo (Hewlett-Packard)
- Cohesion-Digital Equipment Corporation
- HP-Softbench-Hewlett-Packard
- AD/Cycle-IBM
- NSE-Sun Microsystems

One or more of these integration architectures may evolve the become a defacto standard if widespread industry adoption occurs.

CONCLUSION

During the software engineering process, sets of sequential tasks are coupled by a continuing flow of information. In addition, a set of "umbrella" activities occurs concurrently as one sequential task leads to the next. Each task and most activities can be assisted with the use of CASE tools. But the real benefit of the tools can not be realised until the tools are integrated – until other tools can easily use information produced with one tool.

The I-CASE environment combines integration mechanisms for data, tools, and human-computer interaction. Data integration can be achieved through the direct exchange of information, trough common file structures, by data sharing by interoperability, or through the use of a full I-CASE repository. Tools integration can be custom design by vendors who work together or can be achieved through management software provided as part of the repository. Human-computer integration is achieved through interface standards that are becoming increasingly common throughout the industry.

THE ROAD AHEAD

In the previous chapters that have preceded this one, we have explored a process for software engineering. We have presented both management procedures and technical methods, basic principles and specialised techniques, people-oriented activities and task that are amenable to automation paper and pencil notation, and CASE tools. We have argued that measurement, discipline, and an overriding focus on quality will result in software that meets the customers needs, software that is reliable, software that is maintainable, software that is better. Yet, we have never promised that software engineering is a panacea.

As we move toward the dawn of a new century, software and system technologies remain a challenge for every software professional and every company that builds computer based systems. Max Hopper [HOP90] suggests the current state of affairs when he states:

> Because changes in information technology are becoming so rapid and unforgiving, and the consequences of failing behind are so irreversible, companies will either master the technology or die... Think of it as a technology treadmill. Companies will have to run harder and harder just to stay in place.

Changes in software engineering technology are indeed "rapid and unforgiving," but at the same time process is often quite slow. By the time a decision is made to adopt a new method (or a new tool), conduct the training necessary to understand its application, the introduce the technology into the software development culture, something new (and even better) has some along and the process beings a new.

THE IMPORTANCE OF SOFTWARE-REVISITED

The importance of computer software can be stated in many ways. The function delivered by software differentiates products, systems, and services and provides competitive advantage in the marketplace. But software is more than a differentiator. The programs, documents and data that are software help to generate the most important commodity that any individual, business, or government can acquire-information, business and [PRE91] describe software in the Following way:

Computer software is one of only a few key technologies that will have significant impact on nearly every aspect of modem society during the 1990s. It is a mechanism for automating business, industry, end government, a medium for transferring new technology, a method of capturing valuable expertise for use by others. a means for differentiating one company's products.

THE SCOPE OF CHANGE

Change in the technologies that have an impact on computing seems to take on a progression that can be called the 5-5-5 rile. A fundamental new concept seems to move from initiate idea to mass market products in about 15 years. During the first 5 years a new idea is formulated an evolve into a prototype that is used to demonstrate basic concepts. The experimental prototype is refined by scientists and engineers over the next 5 years and the first products (reflecting the new idea) are introduced during this time. The final 5 years are spent introducing the product (and its descendants) to the marketplace. By the end of 15 years (5-5-5), a new idea with technological or merit can grow to encompass a multibillion dollar market (Figure 12). Although the 5-5-5 rule is only an approximation, the 15-year time span from initial idea to major market seems to be a reasonable scale with which we can measure the evolutionary change in the computer business.

The changes in computing over the past four decades have been driven by advanced in the "hard sciences"-physics, chemistry materials science" engineering, The 5-5-5 rule seems to work reasonably well when new technologies are derived from a basis in the hard sciences. However, during the new few decades, revolutionary advances in computing may well be driven by "soft sciences"-human psychology, neurophysiology, sociology, philosophy, and others. The gestation period for technologies derived from these disciplines is very difficult to predict.

For example, the study of human intelligence has been conducted for centuries and has resulted in only a fragmentary understanding of the psychology of thought and the neurophysiology of the brain. However, significant process has been made over the past 30 years. Information derived from the soft sciences is being used to create a new approach to software-artificial neural networks [WAS89]-that may lead to machine learning and the solution of "fuzzy" problems that have heretofore been impossible to solve using conventional computer-based systems.

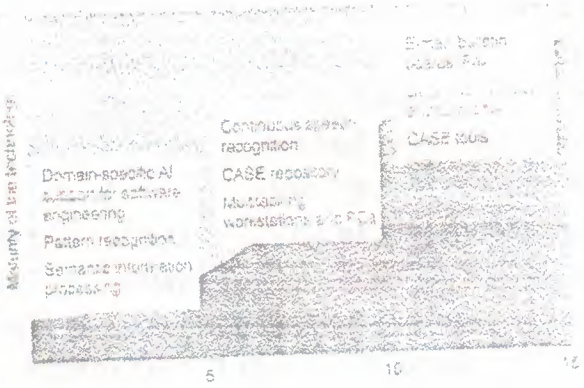


Figure 13. Influences on software engineers and their work.

The influence of the soft sciences may help mold the direction of computing research in the hard sciences. For example, the design of "future computers" may be guided more by an understanding of brain physiology than an understanding of conventional microelectronics.

The changes that will affect software engineering over the next decade will be influenced from four simultaneous directions: (1) the people who do the work, (2) the process that they apply, (3) the nature of information and (4) the underlying computing technology. In the sections that follow, each of these components-people, the process, information, and the technology are examined in more detail.

PEOPLE AND THE WAY THEY BUILD SYSTEMS

A dilemma faces every company that must build computer-based systems in the 1990's. the software required for high-technology systems becomes more and more complex and the size of resultant programs increases proportionally. There was a time when a program that required 100,000 lines of code was considered to be a large application. Today, the average program for a personal computer application (e.g., word process, spreadsheets, graphic programs) is often two to three times that size. Programs build for use in industrial control, computer-aided design, information systems, electronic instrumentations, factory automation and nearly every other industry-capable application often exceed 1.000.000 lines of code.

The rapid growth in the size of the "average" program would present us with few problems if it weren't for one simple fact: As program size increases. The number of people who must work on the program must also increase.

Experience indicates that as the number of people on a software project team increases, the overall productivity of the group may suffer. One way around this problem is create a number of software engineering teams, thereby compartmentalising people into individual working groups. However, as the number of software engineering teams grows, communication between them becomes as difficult and time-consuming as communication between individuals. Worse, communications (between individuals or teams) tends to be inefficient- that is, too much times is spent transferring too little information content and, all too often, important information "fall into the crack."

If the software engineering community is to deal effectively with the communication dilemma, the road ahead for software engineers must include radical changes in the way individuals and teams communicate with one another. In many companies, electronic mail and bulletin boards have become commonplace as mechanisms for connecting a large number of people to an information network. The importance of these tools in the context of software engineering work cannot be overemphasised. With the help of electronic mail or bulletin board system, the problem encountered by a software engineer in New York City may be solved with the help of a colleague in Tokyo. In a very real sense, bulletin boards become knowledge repositories that allow the collective wisdom of a large group of technologists to be brought to bear on a technical problem or management issue.

As hardware and software technologies advance, the very natural of the workplace will change. The following scenario, adapted from Pressman and Herron [PRE91], provides one vision of a software engineer's work environment during the first decade of the twenty-first century.

The environment implied by the above "conversation" will change the work patterns of a software engineer. Instead of using a workstation as a tool, hardware and software becomes an assistant, performing menial tasks, coordinating human-to-human communication, and in some cases, applying domain-specific knowledge to enhance the engineer's ability.

If past history is any indication, it is fair to say that people themselves will not change. However, the ways in which they communicate, the environment in which they work, the methods that they use, the discipline that they apply, and, therefore, the overall culture for software development will change in significant and even profound ways. Recalling our earlier discussion of the 5-5-5 rule, some of the changes that will affect the people who do software engineering work and their current position on the 5-5-5 timeline are noted in figure13.

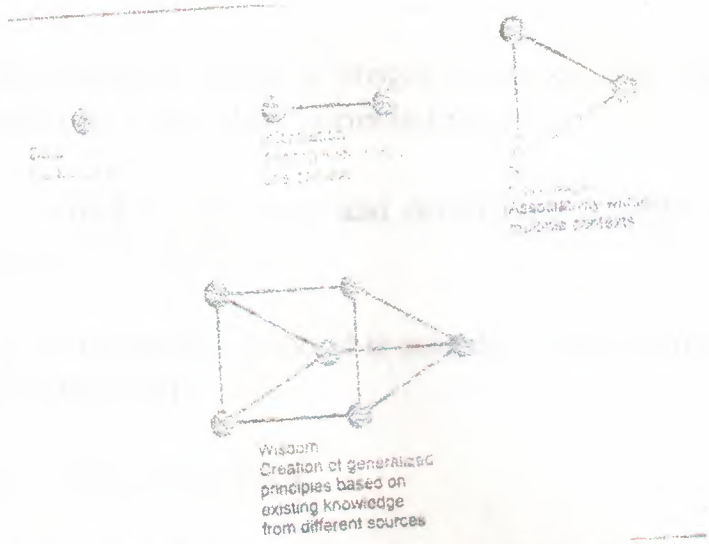


Figure 13. An "information" spectrum.

THE "NEW" SOFTWARE ENGINEERING PROCESS

It is reasonable the characterise the first two of software engineering precise as the era of "linear thinking". Fostered by the classic life-cycle model software engineering was approached as a linear activity in which a series of sequential steps could be applied in an effort to solve complex problems. Although the sequential paradigm for software engineering work can be effective, it is not without the problems discussed in the previous chapters.

The use of the sequential software engineering paradigm will not disappears. A sequential approach will remain effective for those problems in which requirements are well defined, complexity is relatively low, and overall project and technical risks are reasonable well understood. But what about problems that don't fit into this category?

It is likely that a large segment of the software engineering community will move toward an evolutionary model for software development.

The evolutionary model divides the software engineering space into four quadrants? Management planning, formal risk analysis, engineering, and customer assessment. Recalling the discussion of the spiral model, each loop through the quadrants moves the developer closer to a completed system.

With in the engineering quadrant of the evolutionary (spiral) approach, it is likely that the road ahead will have an object-oriented orientation. The object-

ented paradigm for software development offers promise for a number of asons:

- The derivation of reusable program components (classes) is a natural consequence of the object-oriented paradigm.
- Reuse provides immediate and compelling benefits in product quality and process productivity.
- An object-oriented viewpoint is actually a more natural way to examine complex problems.

TECHNOLOGY AS A DRIVER

The people who build and use software, the software engineering that is plied. and the information that is produced are all affected by advances in rdware and software technology. Historically, hardware has served as the hnology drive in computing. A new hardware technology provides potential. ftware builders then react to customer demands in an attempt to tap the tential. Figure 14 applies the 5-5-5 rule in an attempt to place various rdware technologies in the overall evolutionary cycle. Placing a particular hnology on the 5-5-5 curve can be difficult. For example, RISC technology s currently evolved to the product stage, but it is not yet a mature market. ence, it has been placed in the "prototype" stage of technology maturity.

The road ahead for hardware technology is likely to progress alone two rallel paths. Along one path, mature hardware technologies (CISC and RTSC ocessors memory storage, and communications) will continue to evolve at a pid pace. With greater capacity provided by traditional hardware technologies, e demands on software engineers will continue to grow.

But the real changes in hardware technology may occur along another th. The development of nontraditional hardware architectures (e.g., massively rallel machines, optical processors, neural network machine.

We have already noted that software technology tends to react to changes hardware technology. Applying the 5-5-5 rule to software technology (figure 5), the software products of today will be joined and possibly displaced by ftware technologies in the first and second stages of maturity. There is little ubt that the technologies shown in the prototype stage of Figure 15 will come extremely important as the 1990s progress. In fact, object-oriented hnologies may form a bridge between artificial intelligence approaches

inherently object-oriented), conventional software applications, and database technology. In so doing, they may represent an important step toward knowledge processing as discussed in the preceding section.

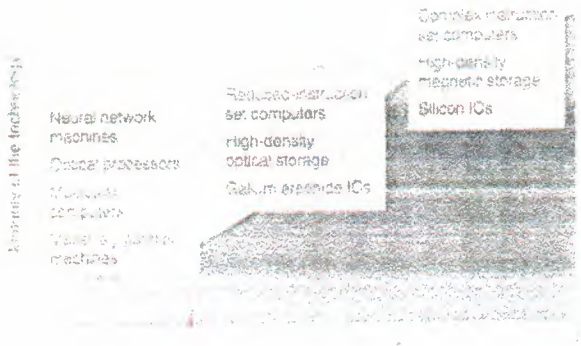


Figure 15. Changes in hardware technology.

The road ahead for software engineering will be driven by software technologies. As software moves more forcefully into the realm of fuzzy problems (AI, artificial neural networks, expert systems), it is likely that an evolutionary approach to software development will dominate all other paradigms. As object-oriented approaches become more prevalent, evolutionary paradigms for software engineering will be modified to accommodate program component reuse. "Foundries" that build "software IC's" may become a major new software business. In fact, as the new century dawns, the today. There may be vendors that build discrete devices" (reusable software components), other vendors that build system components (e.g., a set of tools for human-computer interaction), and system integrators than provide solutions for the end user.

software engineering will change-of that a-e can be certain. But regardless of ho radical the changes are, we can be assured that quality wil never lose its importance and that effective analysis and design and competent testing will always have a place in the development of computer-based systems.

CASE 2000: THE FUTURE OF THE CASE TECHNOLOGY

Computer Aided Software Engineering (CASA) technology has been available to the professional software engineer for more than a decade. Tools to support all phases of the software development lifecycle are commonplace, and most large IT organisations have some form of CASE tool support, whether it be

analyst/designer workbenches, programming support environments, configuration management support or testing tools.

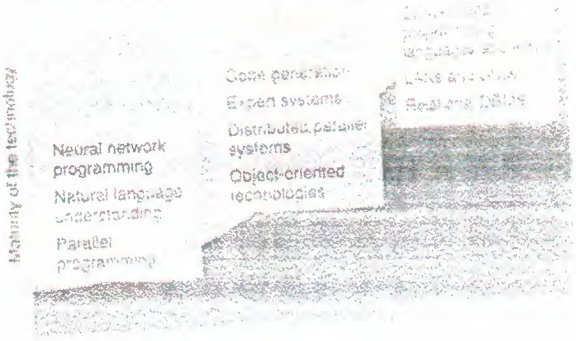


Figure 15. Changes in software technology.

After an early proliferation of tools, a degree of maturity has been reached in CASE technology through the emergence of product leaders, standardisation efforts, improved technology such as integrated CASE (I-CASE) and component. CASE (C-CASE), and an awareness that CASE is part of a wider solution involving people, organisational structures and process.

Several useful references exist, providing a historical context to the development of CASE technology [1], as well as practical insights into CASE products, their use and effectiveness [2-4].

Therefore, as CASE technology stabilises and consolidates, it is an appropriate point to look forward to the emerging themes and trends, and attempt to identify the state of CASE at the end of the decade. To this end, the British Computer Society CASE Group recently organised a research seminar in which participants were invited to present their vision of CASE technology in the year 2000 based on their current research and development. A variety of experts assembled from industry and academia, and during the workshop three areas of discussion emerged.

CASE users

The first theme concerned who should use CASE tools; IT professionals or end users. This was based on the perceived culture gap between the two groups.

It was proposed that it is dangerous to polarise tool users and that there is a spectrum of users ranging from those with very little IT expedience to meta-tool

developers. Each group is likely to need different tool support; ranging from tools for personal information processing to fully featured CASE tools For large scale projects. However, the nature of the software development process is such that end users need to be more involved in the development process. Future CASE tools need to address this issue by providing a range of facilities, empowering the end user to develop sophisticated specifications, and supporting the IT professional as an objective facilitator of user involvement who can integrate user driven systems into the corporate information system infrastructure.

Possible technologies to support this view included intelligent knowledge-based tools, corporate data repositories and advanced human-computer interaction techniques.

CASE process

The second theme looked at the need for organisational maturity before the introduction of CASE technology, and the extent to which such technology could support the entire software development and volution process. Discussion turned to the problem of translation between the informal descriptions, more appropriate to end users, and the IT-oriented models used by system developers. This translation or leap was recognised as involving considerable creativity and would be difficult to reproduce through CASE technology alone. Thus, the idea of automatic translation (i.e. automation of the development process) was rejected in Favour of CASE assisting the human developer.

In looking at the wider issues of introducing CASE, the need for extensive preparation and training before using CASE technology was recognised as a critical factor in the successful and continued use of the tools.

CASE technology

On this final theme, concern was expressed at the inability of current tools to utilise the latest developments in hardware to provide radically different functionality to the user. Developments in multimedia, animation and knowledge based systems needed to be harnessed to improve useability and enumerate user involvement in software development.

However, the point was made that industry understanding of the software development process is still generally poor, and therefore tool development is limited by our current knowledge. Tool integration and object reuse were seen as desirable but not easy to obtain. Integration may lead to "fossilisation" if tool

development fails to keep pace with evolving user requirements. Similarly, object reuse is hindered by the difficulty of identifying and classifying objects.

Overview of the papers

In this Special issue on future CASE, we have selected six papers which cover a broad range of topics within the three themes described above. They range from how the CASE process affects users, through proposed extensions to CASE, to the possibility of re-engineering the software design process to suit CASE.

The papers by Gavin and Little and by Smith review the current use of CASE. Both papers highlight what is considered best practice and propose ways for future wide use of CASE. Gavin and Little investigate the need for CASE within manufacturing information Systems. So often CASE is cited as not meeting its promise and as a disappointment to purchasers. This is said to be so because of the gulf or culture gap between IT suppliers and manufacturers. Gavin and Little describe three detailed case studies, collaboratively undertaken by industry and the Department of Industrial Studies at the University of Liverpool. Differing CASE products were used In each study, typifying the products on the market. Gavin and Little conclude that there is a need for a manufacturing CASE product that can be adapted from existing tools and methodologies.

Smith concentrates on the use of CASE in the software development process. He describes the development, within AT&T ISTEL of a flexible CASE product that can meet the developing needs of different classes of clients, both internal and external. The paper concentrates on how client-centric CASE has been developed and used within AT&T ISTEL to aid their business and software process re-engineering; clients have Included developers of CASE and other software products.

Both Griffiths and Llcyd-Wililams propose the incorporation of knowledge into CASE. Griffiths proposes advancing the concept of analysis workbench to what he calls the third generation. The proposed architecture will capture knowledge of the software process into the workbench. Such an approach, it is claimed, will facilitate the re-engineering of the business process.

Lloyd-Willlams suggests that the Incorporation of domain-specific knowledge into a CASE tool will facilitate the software design process. Knowledge-based CASE enables the tool to play an active part in the design process, offering the user alternative solutions to problems and ensuring integrity. At the same time, design know-ledge can be retained for retrospective use.

Gaskell and Phillips propose linking an executable stage to CASE specifications. The Executable Graphical Specification tool developed at Hull University is described. The excitability Is provided via code in the functional programming language Gofer. The use of a functional language means that rapid prototyping can be achieved. However, there is more work needed to achieve a full implementation. Gaskell and Phillips conclude that excitability needs to be maintained throughout the lifecycle for most development. upgrades and maintenance.

McGinnes takes perhaps the most radical view. He argues that CASE tools were developed to support methods that existed long before CASE. There is a need to develop techniques for design and analysis that are well suited to CASE In Business Process Re-engineering, the idea or using IT to automate existing processes has been abandoned in favour of an interventionist approach. The same needs to be done for Software Engineering; CASE should not just be providing automatic tools for per-forming manually based tasks. New processes should be identified that are possible in a computerised environment. For example, visual modelling can be undertaken, allowing an iterative approach to software development, while hiding technical details. Such an approach would have been impossible in a manually based method.

Towards 2000

Unfortunately, we have been unable to select all the papers we would have wished for this Special issue. We can therefore only present a snapshot of a very small part of the wide range of research and development being under-taken In CASE and its associated technology. Issues such as requirements engineering, support. For soft systems modelling, repository technologies and cooperative CASE have not been explicitly addressed.

However, we believe that the papers presented in this Special issue can act as a contribution to the wider debate on CASE for the year 2000 and look forward to the emergence of even better CASE products.

Finally, the guest editors would like to thank all those who have contributed either to the research workshop or who have submitted papers for this Special Issue. Particular thanks also go to the IEE Publishing Department for achieving publication in such a short period between the original workshop and the publication of this issue.

ciated costs are easy to measure on a financial basis, but the benefits are t to quantify and are often intangible. In order to influence senior manage- nt, information technology has to be seen to address specific Identified tegic needs.

A recent survey by Benchmark Research [3] reveals that investment in nufacturing information technology will rise for the first time since 1989, by 4% or £33m over the next year within the UK, representing an annual expen- ure of £1450 million. The report identifies a low level of satisfaction with rmation Systems that have been implemented in support of manufacturing ning and control. This is due to the unsuitability of the systems and to the lure to deliver the promised benefits. In response to this report, Puttick [4] ted

There seems to be a great gulf between IT suppliers and nufacturers, Manufacturers are so caught up in the chaotic world of the tory floor that they are unable to define their problems adequately, let alone lain them to others. IT suppliers, on the other hand, do not fully understand nufacturing, and are so sales-oriented that they don't focus on the real blem.'

With the expected rise in investment and the proliferation of packaged tems, it becomes increasingly important that manufacturing systems engineers velop a detailed understanding of their organisation's requirements and the blem areas to be addressed. This provides the manufacturing organisation h the knowledge necessary to communicate effectively with software vendors d to select the appropriate solutions. Typically, such solutions will be achieved the selection and tailoring of a range of packages, rather than by the design of espoke software.

The Advanced Manufacturing Systems Research Group at the University Liverpool has been researching the specification and design of integrated rmation systems to support advanced manufacture for a number of years, pported by three major contracts from the ACME (Application of Computers Manufacturing Engineering) Directorate of the SERC. At an early stage in the oup's work, the importance of a clearly understood and unambiguous uirements specification document was appreciated. Such a document should m the basis of any contract between the manufacturing organisation and a tware vendor. To produce such a specification for the complex systems ecessary to support an advanced factory is a daunting task

Traditionally, requirements specification documents for manufacturing information systems have been written in natural language and often portray a 'wish-list' predominantly based on the features of commercially available packaged solutions, rather than the strategic needs of the business.

The complexity of the manufacturing environment dictates the need for a systematic or structured approach to enable manufacturing Systems engineers to analyse and define their information requirements. However, very little advice has been forthcoming regarding this procedure within manufacturing industry. Many authors have identified the need for structured approaches and effective systems modelling techniques, but the few methods developed to satisfy this need have failed to find 'widespread use.

Within the comparatively youthful computer industry, the discipline of software engineering and computer-aided software engineering (CASE) tools have been proven to facilitate the production of reliable, 'well documented information systems that help meet the requirements of the intended users. That is due to the considerable emphasis on the tasks of requirements specification. However, almost ah of this work has taken place outside the manufacturing environment.

If as research at the University of Liverpool suggests, manufacturing Systems engineers are required to analyse their information difficulties and define the associated requirements, it would seem appropriate to utilise established approaches from other application domains, such as the software industry. Although this route is logical and feasible, the methods and tools available for use by the non-specialist (in software engineering terms) manufacturing systems engineer need much development.

This paper presents the research 'work undertaken to Investigate the relevance of software engineering and CASE in manufacturing systems engineering. An in-depth examination of this relevance had not previously been undertaken, and the indications from the manufacturing industry are that CASE has had little real penetration. The research objectives were

- to establish the need for structured methods and tools in support of the production of robust requirements specifications for manufacturing information systems.

- to examine commercially available CASE methods and tools for their suitability to provide support to the manufacturing systems engineer in

developing a specification of requirements for manufacturing information systems.

- to identify the main developments needed in CASE methods and tools to ensure their widespread adoption within manufacturing industry; this formed the basis of the contribution to new knowledge.

An extensive review of literature was conducted into manufacturing information Systems development and the history and current capabilities of CASE methods and tools. In addition, this review examined the tentative use of CASE methods and tools within the manufacturing industry. A survey was undertaken of commercially available CASE products aimed at supporting the specification of requirements to increase familiarisation 'with the CASE market-place and product capabilities. Three typical CASE products, Identified as suitable, were applied on-site to manufacturing information systems projects, This tested the appropriateness of the methods and tools employed within the manufacturing environment.

In addition, experts within manufacturing information Systems development (from both academia and industry) were questioned to gain an external perspective of the needs of manufacturing systems engineering and the relevance of CASE methods and tools. The Initial research hypotheses were examined in the light of the research work undertaken, and conclusions and recommendations were produced for the requirements of CASE for more widespread application within manufacturing industry. It is important to note that this research project was viewed as an introductory study into this area, raising many questions to be tackled in a subsequent project outlined.

We believe that, in order to facilitate a competent definition of a manufacturing organisation's requirements for computer-based technology in support of its strategic objectives, it is necessary for manufacturing systems engineers to adopt structured approaches and tools. As Puttick [5] states.

This mismatch between the Systems manufacturing has installed and what It really needs IS due to an Inability of manufacturing to articulate its needs and a Lack of understanding by the IT vendors... The lack of analysis toots and techniques to translate business and manufacturing needs into if requirements has held industry back.'

Specification of manufacturing information systems

Manufacturing information systems have significantly progressed from the basic accounting and stock control models to the complex integrated systems, embracing materials planning and shopfloor control, found within many factories today. Underlying this evolution has been the realisation of the critical value of timely and accurate Information in support of the manufacturing processes. Due to this, the computer systems supporting manufacturing grew in complexity This complexity was directly related to the volume of information to be processed, as Hollman Berry and Whybark [6] identified:

'Many problems in manufacturing planning and control are not analytically complex; instead their complexity derives fmm the enormity of the underlying database required to properly support routine decision-making systems.'

The analysis and design of manufacturing information systems has been no different to that of information systems operating in other, more traditional, application areas, with similar problems encountered. These problems not only relate to the cost effective production of reliable and maintainable software, but also to the Identification of actual user requirements (which can be particularly complex for manufacturing systems). Both of these problems could be Improved by the use of CASE methods and tools.

Importance of integrated information

In recent years, the importance of the role of information within manufacturing systems has been stressed by many authors [7-11]. The consensus in the literature is that information processing and control is a key area to be considered in developing manufacturing systems. Harrington [10] strongly asserts that information provides the basis for all aspects of manufacturing; manufacturing is, in fact, an information transformation process. The large volumes of information for manufacturing planning and control, and the need to collect, analyse and report on such, create a considerable problem of management.

The critical need is to view' manufacturing in terms of information. Indeed, the way in which to make manufacturing more efficient is to improve on the use of this information and attempt to solve the problems identified also. Manufacturing Information must be appropriate, consistent, timely and accurate. To facilitate this, two elements need to be addressed. First the components, relationships and provision of information 'within manufacturing activities have to be dearly understood by manufacturing systems engineers. Secondly, the

importance of this Information has to be positively appreciated; it is a vital corporate resource. The predominant defect in existing manufacturing systems is the inability to transfer and share Information. Both essential to the support of integration. Halevi [9] argues for an all-embracing' approach to Systems integration, recognising that manufacturing processes are linked and utilising the same information.

The potential of software to support the integration of information across manufacturing systems is now well accepted, and computer-integrated manufacturing (CIM) has become popular in defining the ideal of such an integrated environment [12, 131. The PA Consulting report for the Department of Trade and Industry [14J identifies that the move towards Integrated information systems Is one of the key responses to be made by companies aware of the opportunities and threats facing manufacturing organisations today. The report also notes that few companies make full use of available technology, and fewer make coherent attempts to Integrate their information Systems. Indeed, there is little evidence of a consistent application of appropriate methods and tools within this area.

The importance of the link between integrated information and manufacturing strategy has been observed. The objective may not be to automate the whole of a manufacturing business, but to support those elements of integration that support the defined strategic objectives. This is put succinctly by Evans and Lane (15):

Information is a powerful weapon in the struggle to achieve the frill potential of manufacturing systems. Integration is essential for frilly effective use of the information that exist in different sections of manufacturing companies. Effective use of information impacts upon what can be achieved as well as the way of achieving It.,

The effective use of information could yield a competitive advantage no longer possible simply by the efficient use of advanced production technology. Information could help management address the conflicting pressures of cost, customer service, manufacturing flexibility and time to market However, Bollingum [11] presented the following caveat:

'Manufacturing industry's track record in information management is not good, and as the opportunity and competitive pressures to move towards computer integrated manufacturing mount, we must think strategically about what we want information for and how we will use it to manage our business.

The level of investment predicted in factory *information indicates that the costs of failure are going to be high.'*

Need for a structured approach

We have observed that the effective processing and integration of information is key to an organisation achieving competitive manufacturing, and that software is essential In supporting this. However, the trend towards more complex information systems handling increasingly dynamic information reinforces the problems outlined above. This situation has prescribed the need for a Systematic or structured approach to be applied in designing and implementing manufacturing information Systems. This need was confirmed by Cassidy et *al.* [6] as part of the IEEE task force on research needs in manufacturing systems. They saw that no Systematic method for design and implementation was currently available, and that more research and development "were needed on providing effective approaches and modelling techniques for manufacturing systems engineers.

We have [17] found that, although there was general agreement on the need for a structured approach, there was very little advice forthcoming. Few companies employed such approaches, and frequently the integration of the manufacturing information system under development 'within the existing factory systems was not considered at all or left until too late:

The system software will be largely developed and systems designers are unlikely to be willing to make the *major amendments at a later date. Another "island of automation" has been created.*

Hitomi [7] has defined manufacturing Systems engineering as the use of a systematic approach within the manufacturing environment Wu [18] was particularly surprised that although there was a great deal of literature on the techni cal aspects of advanced manufacture, there was very' little regarding structured approaches to design and implementation.

Two modelling techniques, GRAI (Graphe a' Resultats et Actividés Interlies) and IDEF (ICAM Definition), featured highly within the reviewed literature. Both were developed specifically for the manufacturing environment.

The GRAI approach was developed by the GRAI Laboratories of the University of Bordeaux for analysing and designing manufacturing planning and control systems [19-21]. Two graphical modelling techniques are used, the

GRAIgrid and GRAInet, for depicting decision making and functions supporting decisions respectively. Fig. 16 shows these techniques.

The IDEF set of graphical techniques were developed by the US Air Force and collaborative organisations as part of the CAM (integrated Computer Aided Manufacturing) Programmer. Three techniques are utilised and are presented in fig. XXX. IDEF provides a functional model and is a subset of SADI. IDEF provides information modelling and is similar to data model techniques used in software engineering. The third technique $IDEF_2$ provides modelling of the dynamic aspects of a system based on simulation technique.

Although many differing approaches have been developed for the manufacturing environment; including GRAI and IDEF, they are still not In widespread use. Whatever the reason for this, the problems associated with designing and implementing manufacturing information Systems remain. As Naylor and Votz [24] state.

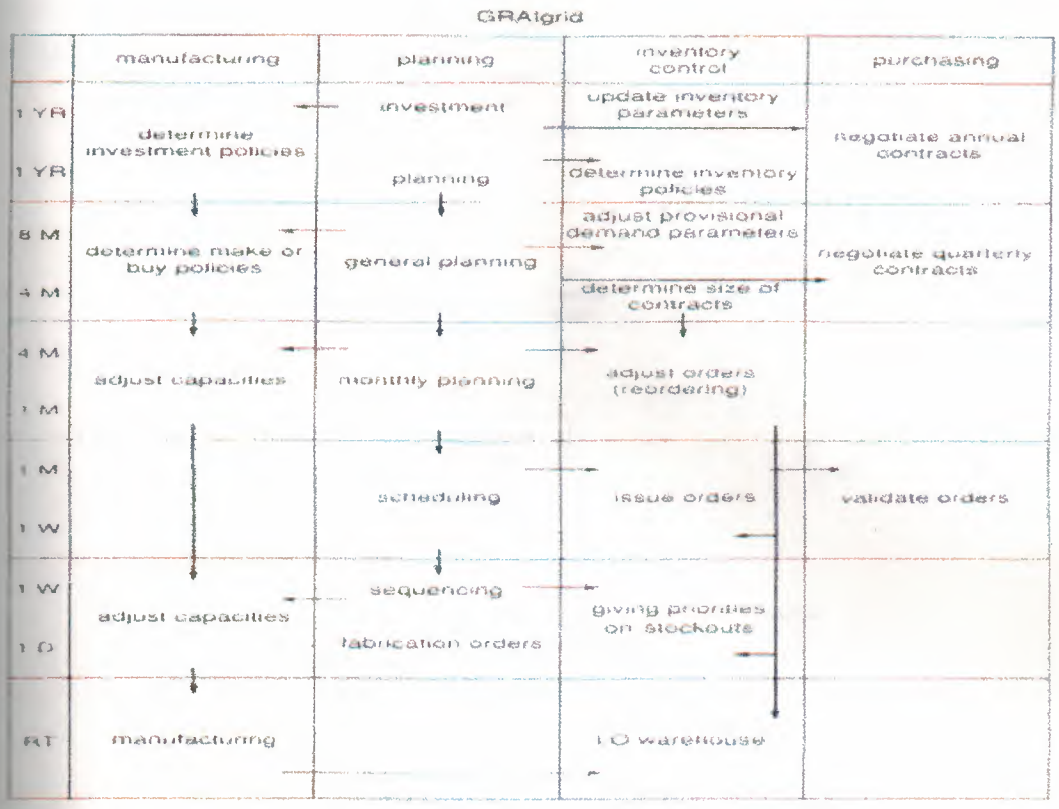
Arguably, software is the integrated manufacturing problem. The machines, robots, material transports, and so forth exist, but the software needed to tie them together into orchestrated flexible robust systems does not.

CASE within manufacturing industry

Whereas manufacturing industry has assessed the importance of information and of the need for structured approaches and modelling techniques in identifying manufacturing system requirements, the software industry has established the discipline of software engineering, with automated support, to help in satisfying these needs for its own systems. Such approaches provide an effective approach to developing information systems. Therefore, it is evidently not surprising that the approaches derived for this discipline are slowly finding application within the manufacturing environment.

One of the aims of the Alvey Programme's GEC 'Design to Product' Demonstrator was the promotion of software engineering within manufacturing [25]. Data flow diagrams were utilised within the CAM-I Factory Management Project [20], initiated to demonstrate the concepts required for advanced factory management They' have also been pro-meted within development workbooks as a high-level approach to mapping the information flow 'within a manufacturing organisation.

Other authors [18, 26-28] have asserted that the discipline of software engineering is a key technology' for implementing computer-integrated manufacturing and have promoted the use of data flow diagrams, in addition to IDEF, as fundamental development techniques.



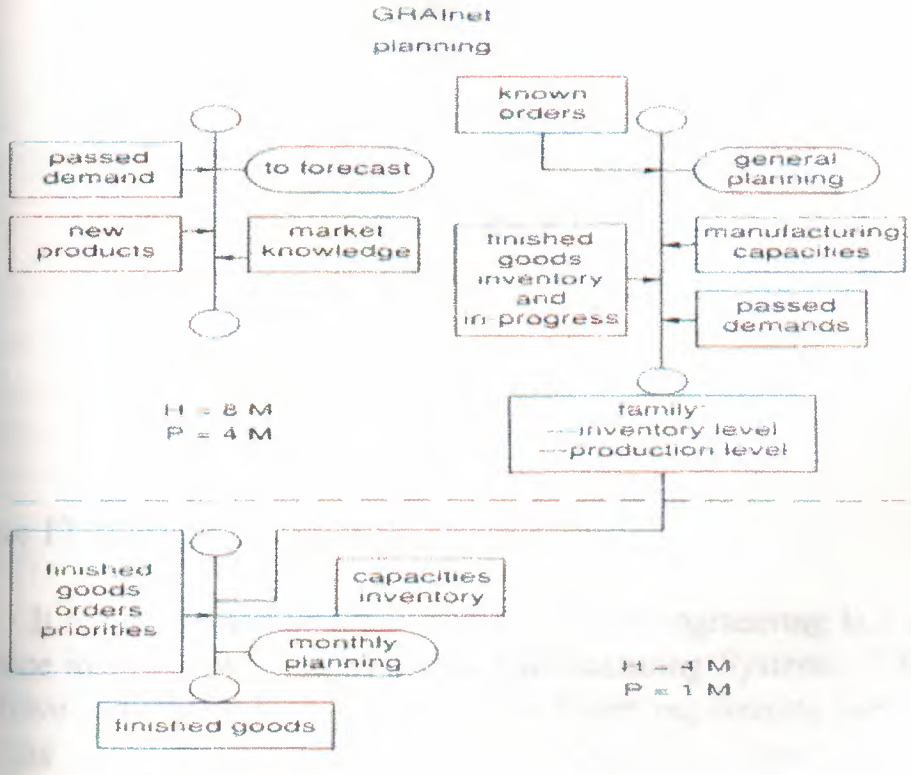


Figure 16. GRAI modelling technique.

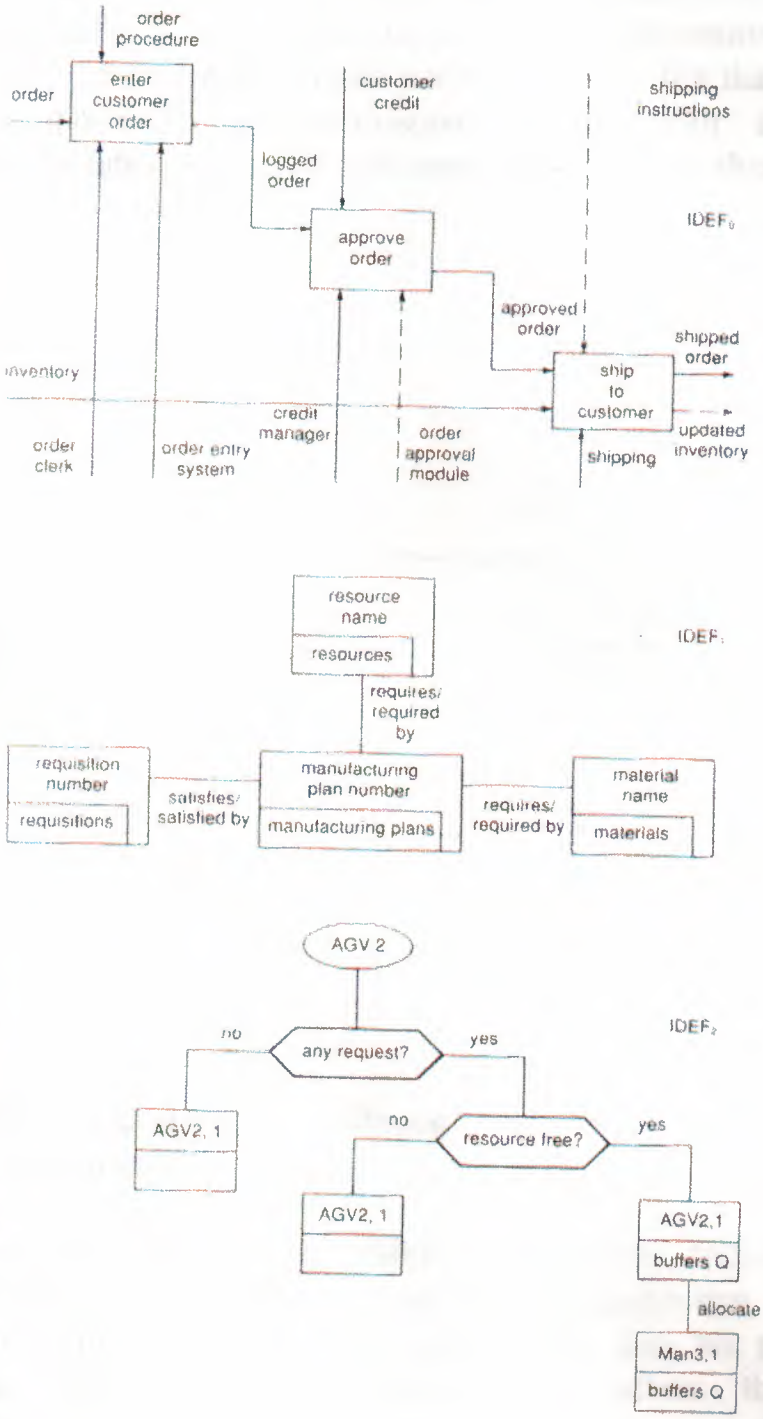


Figure 17. IDEF modelling techniques [22,23].

It is our opinion that the use of software engineering is a necessary practice in developing the advanced manufacturing Systems of the future. We have actively promoted the use of software engineering methodologies and the associated techniques. These approaches have been used to

provide the basis for effective implementation of plant management Systems 'within collaborating manufacturing organisations. Despite not being specifically developed for manufacturing information systems, the use of CASE offered considerable potential. It was felt that the need for integrated manufacturing Information systems with their inherent complexity dictates the use of software engineering methods and CASE tools.

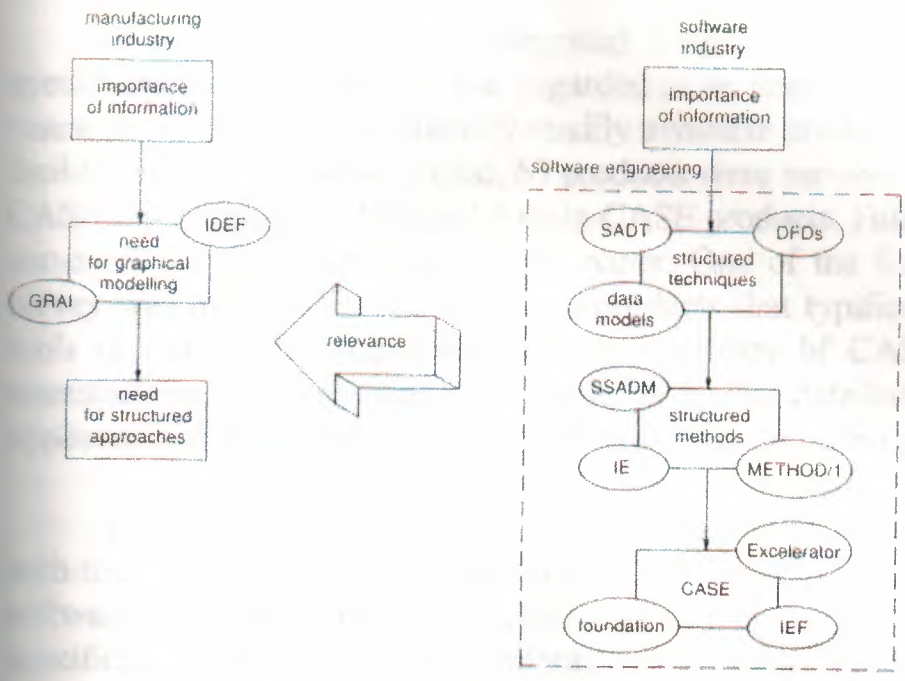


Figure 18. Relevance of software engineering and CASE to manufacturing industry.

The dominance of the GRAI and IDIEF techniques within manufacturing industry is derived from the assumption that, as they were developed within the manufacturing environment, they are possibly more appropriate than software engineering approaches; that there is apprehension in applying techniques from a different application area. However, software engineering provides comparable techniques to GRAI and IDEF 'within a more rigorous framework and with the additional benefit of commercial automated support.

The foundation for the research work undertaken here is that such software engineering approaches and CASE tools are relevant to manufacturing Systems engineering, and they offer considerable potential

in the development of manufacturing information Systems, Fig. XXX attempts to represent this relevance. If more effective Information systems development strategies within manufacturing systems engineering are to emerge, analysis is required of the approaches offered by software engineering and the utilisation of current software development technology; 'with the consideration given to the requirements for methodologies and tools more appropriate for manufacturing. The research presented in this paper represents a first step towards this goal.

Application of CASE within manufacturing industry

A survey of front-end integrated CASE tools that support the specification of requirements was regarded as an important element of the research; the aim being to identify readily available products and the main facilities offered by such. In total, 63 products were surveyed: 48 front-end CASE, 11 integrated CASE and 4 meta-CASE products. Full details of this survey are beyond the scope of this paper. One of the functions of the survey was to identify several leading products that typified the range of tools on offer. The examination of the suitability bf CASE within the manufacturing Industry was then extended by the detailed study of the application of three representative tools In current industrial projects.

The objective of the use of case studies was to increase familiarity with the facilities offered by three of the main CASE products available to software engineers and, by actual use in developing requirements specifications for manufacturing information systems within collaborating companies, evaluate their potential impact The case studies were *situational,* determining a problem within a soft context and utilising CASE in an attempt to solve it; *collaborative,* working 'with project engineers 'within the manufacturing organisations; and *participatory,* as we were involved in the application of CASE.

The projects were defined by the companies involved, on the basis of real and pressing needs at that point in time. Senior management within the manufacturing organisations participated in the projects, affecting and controlling the developed solution.

Two front-end CASE products and one integrated CASE product were selected for the case studies. They were matched 'with an appropriate project by the relative size and capabilities of the tools The front-end CASE tools, AUTOMATE PLUS and Execrator, were methodology-dependent and generic, respecify, and represented typical tools for analysis

and design. The integrated CASE product, FOUNDATION, provided the greatest range of functions of the tools surveyed. As 'with most Integrated CASE tools, FOUNDATION is methodology-dependent

Case study one

The first case study Involved the application of LBMS AUTO-MATE PLUS Front End CASE tool in support of the SSADM methodology, In a review of computer Systems at a medium-sired plastics injection and moulding company. The maintenance of reliable deliveries to customers is critical to the success of the business To this end, the company utilises advanced technology processes and equipment to place Itself among the top five UK producers In a market of over 2000 competing organisations. There has been extensive Investment in computer control and monitoring of the manufacturing processes, Including computer-controlled presses with robotic arms for transferring mouldings from the presses to packing stations.

Senior management were aware that the market-place offered considerable opportunities for expansion and diversification However, a number of problems hindered the company. In moving towards this intent the case study project, as requested by senior management was a review of existing systems for manufacturing planning and control, leading to the soft caftan of requirements for improved systems. In particular, the manufacturing control package in use had met the needs of the company, but lacked the depth of facilities required for capacity management and long-term production planning. Many of the adopted procedures and conveniences reflected inadequacies within the control system and the company's means of overcoming them.

The SSADM documentation was used in the phased advancement of the company as an Information technology-based manufacturing organisation requiring centralised database systems. A Systems manager was appointed to oversee this development New hardware was Installed that could easily accommodate any likely growth over the next five years. During the case study, the user awareness fostered by the utilised approach was prominent and noticeable. Senior management were particularly impressed with the overview provided.

Case study two

The second case study Involved the application of INTERSOLV's Execrator Font End CASE tool. The study concerned the development or a requirements specification for a cutting tool management information system within a flexible manufacturing system (FYLS) of a high technology manufacturing organisation, a privately owned manufacturer of drive lines, axles and transmissions for the special-1st agriculture and construction vehicle market. The company is renowned for its image of utilising advanced manufacturing technology, particularly in flexible manufacturing and materials handling systems. Considerable investment has been made In computer support for production. This includes automatic guided vehicles CAGVs), automatic warehousing systems, dedicated machining cells and a flexible manufacturing system.

The rapid development of the factory has placed considerable strain on the information systems planning and controlling manufacture. Senior management had identified that tool management 'was an area of factory control key to the improvement of production performance, particularly within the flexible manufacturing System. Tool management is concerned with the adequate supply of cutting tools to the machining centres, thereby maintaining the production schedules. Put simply, the right tool has to be at the right place at the right time and with adequate unexpired cutting life. Tooling requirements are dependent on the product design and materials, and therefore will vary from simple to complex depending on the actual parts to be machined.

In this organisation, tool management issues had been ignored for the most part, and this had created a number of Systems operation problems. Most manufacturing work was planned with the assumption that tooling will be avail-able when requested; this is not always the case, resulting in excessive cutting tool inventory.

Tool management is traditionally a manual activity, but automation of the supply of tooling is seen as mandatory within a CIM environment in order to contend with the requirements quickly and effectively. My tool management system characteristically consists of a set of tool stores and tool magazines on the machining centres, a tool handling system and ancillary activities such as tool presetting and tool inspection. This can be upgraded to include automatic tool transfer from stores to machine, and *vice* versa. It was recognised by senior management that the manual tool management system was far from adequate in planning and controlling the

tooling requirements for the FMS. The company needed an effective information System.

By applying CASE, the project objective of generating a requirements specification for a computer-based tool management system was addressed effectively. The development of this specification enabled the manufacturing Systems engineers to gain a lull understanding of the tooling activities involved. A full review of all commercially available tool management information systems was conducted, with each package evaluated against the criteria of the specification document

The tangible benefits from the Installation of the chosen System have been a saving of 5% in tool expenditure per year, a reduction of 10% on machine down-time caused by tooling-related problems, and a reduction in scrap of 10%. Due to these savings, the cost of the system was recovered in less than six months.

Case study three

The third case study involves the application of Andersen Consultant's Integrated CASE toolset FOUNDATION and the corresponding METHOD/1 methodology. This study concerned the selection of a packaged system to plan and control cellular manufacturing at an electrical equipment manufacturing organisation. At the time, the company concerned was considering the reorganisation of Its fabrication shop in line with the implementation of a flexible sheet metal cell, and agreed to allow the methodology' and tool to be used to identify the requirements for manufacturing organisation and control systems.

Over the past few years, the company has made considerable Investment in new manufacturing technologies such as CAD and CNC. In addition, great effort has been applied to implementing corporate business systems, such as MRPII, In order to plan manufacturing more effectively. However, it became apparent to senior management that the existing shop-floor control and reporting mechanisms were far from adequate, suffering from fragmented responsibilities, Inadequate communication and inaccurate manufacturing information.

The critical requirement was the introduction of a new manufacturing philosophy advocating a manageable and flexible factory organisation, but employing strict control over the manufacturing

operations. There was also a need for management to be able to see the current status of orders and machine use on the shop-floor.

The case study was initiated with the objective of defining a control system for the fabrication shop. In order to schedule and control the available resources to meet the required production output. During the early stages of the project, it was decided that this. System of control should be based on the philosophy of cellular manufacturing. The intention underlying the project was that the fabrication shop would become an autonomous mini-factory, per-forming its own related manufacturing functions and comprising work centres grouped Into cells, with each cell producing parts with similar attributes.

The shopfloor management require accurate information not only to run the distinct areas as businesses, but also to provide corporate management with an under-standing of the true state of the factory; hence the need for distributed computer power in the form of supporting control systems. The typical hierarchy of control is one based on area and cell control systems. All related manufacturing operations within the factory are grouped into areas. The factory-wide system, using MRPII, Is responsible for establishing a basic production schedule for all areas of the factory. The area control system allocates its avail-able resources on this basis to areas with a planning horizon of weeks, and reports back to the corporate level on such maulers as completed work orders, stock changes and production performance. The actual manufacturing processes are then carried out at cell level within a planning horizon of hours to days. The cell control system sequences the jobs through the operations, scheduling and controlling the resources of that cell, monitoring and reporting on the progress and work completion. Hence, each area becomes an autonomous mini-business, planning and controlling the specific manufacturing functions to be executed at cell level. Fig, 18 presents the typical hierarchy of area and cell control within cellular manufacture.

By applying METHOD/1 supported by FOUNDATION, the project objective of selecting a control system for the select metal fabrication shop was attained in a highly competent and planned manner. The project was viewed by senior management as a considerable success in developing an approach to manufacturing organisation. The Finance Director referred to the project in an internal newsletter as generating a CIM (Computer Integrated Manufacturing) strategy' to take us to the end of the century.

On the basis of the project results1 a manufacturing system engineering department was formed to implement the defined strategy using the chosen software. The first cell, comprising CNC turning centres, has recently been completed, and the fabrication shop Is currently under development The company now boasts, in an international advertising brochure, of the introduction of cellular manufacture 'allowing greater flexibility, a cut in manufacturing lead times and more simple control of quality'.

CONCLUSION

The application of three typical CASE products to the analysis and specification of requirements for manufacturing Information systems provided tangible 'hands on' familiarisation with CASE, and tested the appropriateness of the methods and tools employed.

All of the case studies attained the objectives set by senior management in an effective and structured manner. Indeed, the studies 'were considered as particularly successful by the organisations involved, and the impact of the methods and tools is most notable within the FOUNDATION case study, providing the basis and justification for revolution using the manufacturing organisation and control approaches.

We believe that the case studies demonstrate the relevance and considerable impact of software engineering methodologies and CASE tools on the specification of requirements for manufacturing Information systems. We consider Execrator to be the superior product of the CASE tools applied, and if coupled with methodology management and project planning and control facilities similar to those of FOUNDATION, it would provide a sufficient cornerstone for a Manufacturing CASE product

Questioning of manufacturing experts

In addition to the work outlined above, It was deemed important to gain an external perspective on the suitability of CASE within manufacturing by examining the views held by selected experts within academia and industry. To achieve this, a modest survey of notable authorities involved with manufacturing Systems development was designed implemented and appraised. Although the sample was, of necessity, small, the opinions and insights cliched helped to focus and strengthen the conclusions of the research topic The information sought from manufacturing professionals

substantiated many of the Issues emphasised earlier and brought a broad perspective to the research work

The coarsenesses view of the survey participants is that CASE methods and tools are appropriate in supporting the specification of manufacturing information system requirements, but that the techniques and tools should be orientated towards manufacturing, simple and easy to use. Considerable planning is needed for the introduction of CASE, with proper estimation of the effort required, the selection of appropriate tools and a supporting organisational environment The important consideration. Is that manufacturing personnel should undertake the s-fl-cation tasks, utilising their valuable knowledge of the operating environment. Any structured approach should empower the end-users in the analysis and design of information systems.

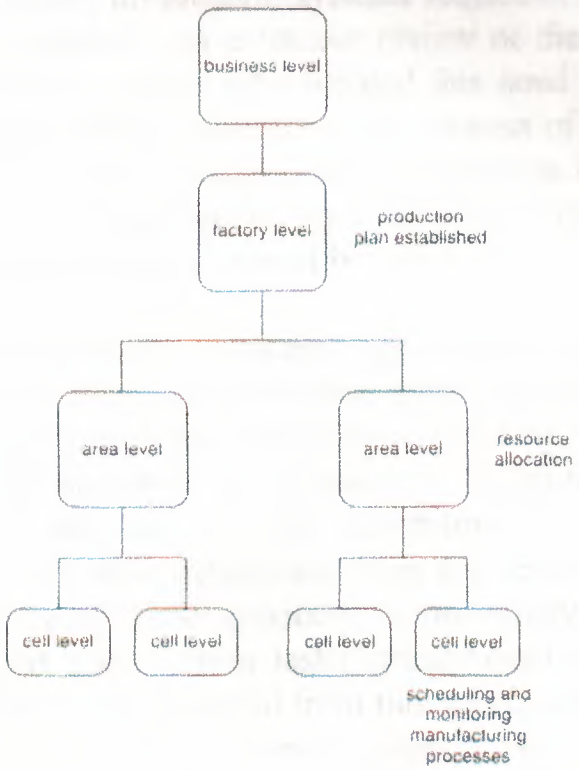


Figure 19. Hierarchy of area and cell control.

There is evidence that the manufacturing industry is starting to adopt software engineering methods and CASE tools. However, a thorough investigation required to determine the needs of manufacturing. System engineers and correlate them with the capabilities of CASE approaches. In

addition, the capabilities and benefits of adopting CASE methods and tools have to be brought to the attention of manufacturing systems engineers, with documented examples of the Improvements made.

Conclusions

The aim of the research presented here was to investigate the need for structured approaches and tools within manufacturing information Systems development, and the relevance of established software engineering methodologies and supporting CASE tools in meeting this need. The research topic has focused on what is widely seen as the most critical task within information systems development, the specification of requirements.

The first objective, the need for structured methods and tools in support of manufacturing information systems requirements specification, has been clearly established. An extensive review or the literature and a survey of manufacturing experts substantiated this need and the requirement for graphical modelling techniques. The success of the case studies within the collaborating manufacturing organisations in comparison with their traditional *ad hoc* approaches also confirmed that a disciplined approach with automated support was of benefit.

Commercially available software engineering methodologies and CASE products were examined for their suit-ability in providing support to manufacturing systems engineers In the task of requirements specification. This was the second objective of the research. To gain familiarity with CASE technology, the history and capabilities of both software engineering and CASE were established from the reviewed literature. A comprehensive survey of CASE products, commercially available within the UK, that support specification tasks strengthened this examination. Three leading products were selected from this survey and applied on-site to manufacturing information systems projects, to gain a working familiarisation with CASE In a real and immediate situation to validate their suitability for the manufacturing systems engineer working within Industry.

To address the final objective, the main developments required for both CASE methods and tools and the sup-porting manufacturing Systems engineering organisation have been identified to facilitate the widespread application of CASE within this area. This has been undertaken on the

basis of the literature review, use of existing CASE methods and tools, and views elicited from manufacturing experts and personal experience.

Here we present the conclusions drawn from the work undertaken and the main developments required if CASE is to find widespread adoption within the manufacturing industry. Substantial further research is outlined that will build on the results of this work

Discussion

The first hypothesis underfring this research work is that structured methods and tools are needed in manufac turing information Systems development particularly in the definItion of user requirements. This premise has been firmly demonstrated by this work Computers supporting manufacturing planning and control are now viewed as a necessity, rather than a luxury. Manufacturing information is complex and typicailly large in volume. Software is needed to process this information, and it has to be robust and effective. The developed Systems have to promote Systems integration, which is accepted by many writers as an aim In itself, and meet the strategic requirements of the manufacturing organisation. This is a pressing Issue for the late I 990s.

To meet these characteristics of computer-based applications, manufacturing Systems engineers have to understand the role of information and the supporting software not only to realise the potential of the manufacturing pro cesses, but also to communicate 'with Systems vendors and select appropriate information Systems. The high level of investment implies that casual and unsystematic approaches are not adequate to competently design and fully justify such systems. The large costs involved means that them can be no allowance for errors.

The use of graphical modelling as a basis for a disciplined approach has been asserted. The analysis of complex manufacturing Systems needs diagramming to reduce the associated complexity and to improve communication of the requirements. To facilitate this in a productive and consistent manner requires the use of automated support Manufacturing Systems engineers require a structured and professional approach to investigate their Own problems, and to simplify and accelerate the development of manufacturing information Systems.

The second hypothesis was that the basis for structured methods and automated support for manufacturing Systems engineers currently exists In

the form of software engineering methodologies and CASE tools. The subsequent assumption is that as the discipline of software engineering has arisen from a different applications area, considerable enhancement is necessary to accommodate the systems development requirements of the manufacturing environment

At a high level, the relevance of software engineering and CASE to computer-based manufacturing planning and control seem obvious; manufacturing Information Systems are after all information Systems. Techniques and approaches developed to design and implement Information systems would appear to be applicable. However, the manufacturing environment requires software to coordinate, control and integrate machines, material, labour and Information, in real-time. The Information Systems required of manufacturing systems differ from the more traditional batch-orientated data processing Systems that CASE was developed to address. In traditional applications, Information is the only consideration. In manufacturing applications, the role of information has to be considered alongside more physical issues.

Manufacturing systems engineers require methods for defining information System requirements preferably based on graphical modelling. Software engineering approaches meet this need, with structured techniques that are predominantly graphical. Software engineering and CASE have been proven to facilitate the production of reliable, well documented and quality software, with improvements in development productivity and maintain-ability. The main weakness of CASE relates to the lack of product integration and the limited functionality on offer, particularly the poor diagramming facilities. In addition, a long learning curve and 'user-unfriendliness' detract from the benefits. To be successful, the implementation of software engineering and CASE has to be carefully planned and supported.

The application of CASE products in the manufacturing industry presented in this paper has validated the relevance of CASE to manufacturing information systems development The case studies were very effective and successful. The employed methodologies introduced systematic approaches to the organisations to deal with problem analysis and information Systems justification. The use of graphical techniques had a prominent and noticeable effect on the understanding of the personnel involved. The developed knowledge improved negotiations with the software vendors.

The CASE products used in the case studies competency supposed the techniques dictated by the methods. However, the difficulty in use was aggravating, and the diagramming facilities were limited and irksome. The reports generated by the tools were more appropriate to the analyst and not suitable to include -within management reports. It was also felt that the support of a specific methodology by a CASE product restricted the customisation of the method by an organisation.

The relevance of CASE was supported by the views of manufacturing experts, but concern was expressed regarding the level of suitability to manufacturing systems and the ease of use. To a few participants, the implementation of CASE was an intimidating prospect. A detailed investigation was requested to examine the appropriateness of CASE in full, defining the specific needs of manufacturing systems engineers and correlating with the facilities of software engineering and CASE.

Computer scientists have imitated manufacturing industry by adopting engineering principles to software development and mimicked the role of computer-aided design technology to develop CASE. Due to the increasing importance of computers within manufacturing, manufacturing systems engineers must now adopt these software engineering principles and CASE technology to realise ambitious and competitive production:

The assumption that the development of CASE is needed to align with the requirements of manufacturing systems engineers has been addressed. The necessary developments required of software engineering methodologies and CASE products are outlined below. These are to be addressed. In detail by a research project outlined. It is interesting to note that the necessary improvements to CASE are not as great as was envisaged at the beginning of this research. The emphasis is on the methods and the organisation required to support the implementation of CASE in manufacturing information systems projects. We suggest that it is not the fundamental nature of the CASE approach that will need considerable advancement but the expertise and background of the potential CASE user within manufacturing industry, the manufacturing systems engineer. Nevertheless, Improvements to the methods and tools will help to bridge the cultural gap.

Requirements of CASE in manufacturing

An important point that we did not fully appreciate at the outset of the research, and that has been firmly emphasised by the work undertaken, is that in the application of CASE, the software engineering methodology is the foremost concern. The method dictates what should be accomplished, what structured techniques are required etc. The role of the CASE tool is to assist the software engineer in the implementation of the methodology; CASE is computer-aided *software engineering.* If software engineering methods and CASE tools are to be adapted to match the requirements of manufacturing systems engineers, the methodology would be the focus of attention. The basic functionality of the supporting CASE product would not differ in any great respect, with the exception of the structured techniques and documentation to be sup ported, and issues influencing the ease of use.

The determination from the research undertaken and the views of manufacturing experts is that, to be of suit-ability to manufacturing information Systems development, software engineering methods should be oriented towards the manufacturing environment. The approaches should dispense with computer science jargon and use language that manufacturing Systems engineers can understand. The methodology should deal with manufacturing-related information issues, which appear to be more complex than typical software engineering applications, and distinctly deal with physical considerations. In addition, manufacturing Systems engineers should be educated to produce hybrid manufacturing and information systems engineers. This would initially appear to be inadequate in time and cost, and possibly detract from the main concern of manufacturing Systems engineers, that of producing manufacturing Systems to facilitate effect production; but this Is a vital issue to be considered in the education of the manufacturing engineers of the future.

A manufacturing-oriented information systems methodology should be rich In organisational context, providing a tangible business foundation for the application of the more abstract graphical modelling techniques. Standards for documentation, particularly the important specification of requirements, should be defined and examples provided. The method should discuss systems development in the real terms of the business, the manufacturing users and the packaged systems market-place.

The implementation of an information Systems development method is a large task in itself and project planning and management Is needed, It

is preferable that such project control tasks are integrated Into the method to ensure that the manufacturing systems engineer addresses these tasks and the appropriate reviews, quality assurance and signoffs are performed.

The amount of documentation required by a methodology should not be excessive. For small to medium-sized development projects, the method could be directed at the production of one document, the requirements specification, with each step aimed at generating sections of this document. The larger methodologies such as METHOD/1 are essentially performing this; at each step, a number of documents are created to be consolidated into significant documents at the end of major development stages.

However, it is often difficult to grasp the overall view in the midst of such a lengthy development process.

It is also important to consider how the methodology to specify the manufacturing information system would integrate with the specification of the other issues to be considered when developing a manufacturing System. An overall manufacturing Systems engineering methodology would address the information Systems concerns and the physical issues such as the means of control, the manufacturing processes to be implemented and the organisational layout of the system. Evidently, these issues influence each other in the specification and design of a manufacturing system. fig. XXX attempts to highlight the issues involved in such specification. A manufacturing Systems engineering method that addresses manufacturing strategy In line with corporate objectives and facilitates the redesign of the manufacturing organisation would merit further research.

We envisage that a CASE product conceived for manufacturing information Systems development would not be too different from existing tools with respect to the basic functionality the provision of diagramming, the design database, model checking and reporting facilities. A definition of the detailed requirements of the manufacturing systems engineer would determine what structured techniques and documentation are appropriate and should be automated within 'manufacturing CASE' products.

The main requirements of CASE emerging from this research relate to the ease of use, particularly the user interface. To be of benefit to manufacturing Systems engineers, CASE products need to be simple to use and easy to learn. The tools have to be obvious and intuitive, and the

functions provided by the tool should be self-explanatory to the user. It is important that both the notion of CASE and the tool itself are not intimidating to the manufacturing Systems engineer. The tool should be Fig- 20 Requirements of software engineering and CASE based on a few simple concepts that are easily understood. The functionality should be partitioned and presented In a helpful way, yet be flexible enough to cater lor both new and experienced users.

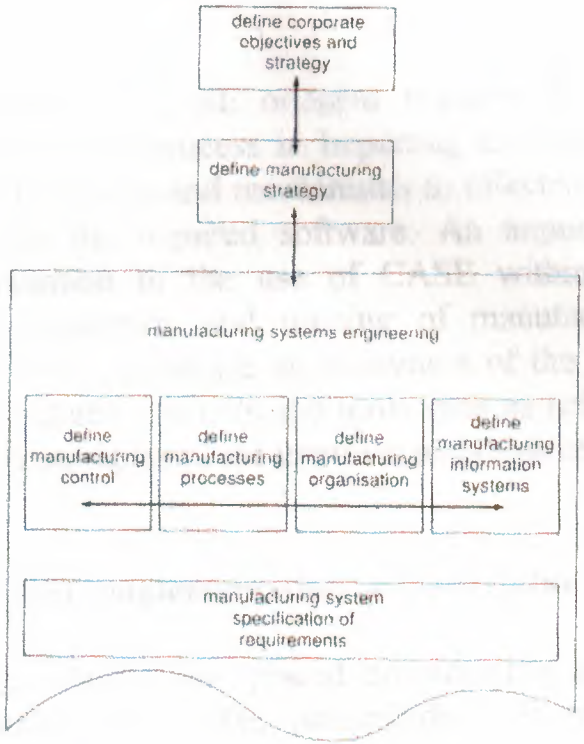


Figure 20. Specification of manufacturing system.

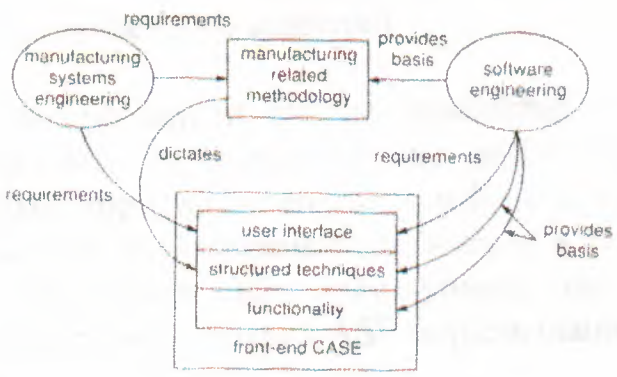


Figure 21. Requirements of software engineering and CASE.

Fig. 21 presents an overview of these requirements of CASE in manufacturing, based on the work undertaken here. It would seem indisputable that software engineers utilising CASE in more traditional application areas would also desire changes to the tools concerning ease of use. The development of an 'ideal' software specification and development environment would be of benefit to both the manufacturing and software industries.

The development of CASE oriented towards the manufacturing environment is a significant process in Imparting manufacturing systems engineers with the knowledge and mechanisms to effectively analyse their problems and specify the required software. An important supporting element is the promotion of the use of CASE within manufacturing industry. Ostensive education and training of manufacturing systems engineers are required to encourage an awareness of the capabilities and benefits of using structured methods and tools such as software engineering and CASE, -with examples of the improvements that can be made.

Requirements for CASE implementation in manufacturing

The software industry has placed considerable emphasis on the planned implementation of CASE, particularly following considerable publicity regarding failures. Manufacturing systems engineers must take note in order to utilise CASE successfully in the specification of manufacturing information systems. To facilitate the implementation of CASE, the manufacturing industry must exhibit the same characteristics as successful software development organisations. Attention must be paid to the organisational environment supporting the use of CASE, and an implementation strategy should be employed.

The organisational support for the introduction of CASE is an important consideration. To utilise software engineering and CASE themselves, manufacturing systems engineers will need ownership of the methods and tools. For small organisations 'without a centralised information services department, this may present no problem; the manufacturing engineers will initiate CASE implementation and be fully, responsible for this.

However, in larger manufacturing organisations with centralised information services or the more traditional ownership of information technology by the financial functions, the organisational structure has to be an important factor in the CASE implementation strategy. It would not be possible for manufacturing Systems engineers to utilise CASE methods and tools without regard for the corporate IT strategy. An alliance must be formed between the manufacturing and information services, and an appropriate organisational scheme adopted.

For the effective Implementation of CASIE, manufacturing systems engineers must have a strong commitment to the use of structured approaches and tools, supported by encouragement from the manufacturing management. Considerable training is required to foster this commitment, and the effort required for implementation should not be underestimated. A CASE champion is required to provide the inspiration and enthusiasm for the application of the methods and tools.

Strategies for the successful Implementation of CASE have been presented within the software engineering literature [29-31]. It is important to note that these strategies do not consider the organisational support for CASE, presumably because software development for more traditional applications is isolated within the information services department. In addition, such strategies define the implementation of a software engineering methodology and supporting CASE tool within one pass, utilising both within a pilot project. However, for a manufacturing organisation, we believe that a two pass implementation strategy is required as two important and transforming approaches are introduced. The first pass of an implementation plan must consider the introduction, utilisation and evaluation of the software engineering methodology, and the second pass undertakes the same considerations -with respect to the supporting front-end CASE tool.

Further research

A number of possible future research programmes became apparent during the execution of the research, The most prominent research need is to broaden the investigation of software engineering methods and CASE tools -within the manufacturing industry, to include the entire software development life-cycle and the widest definition or CASE. This may not be an elementary topic. Although methods and tools for requirements specification have matured and become established, support for the other stages

of the development life-cycle is still in a state of significant fluctuation and evolution.

Two research topics have been outlined above. The first entails the definition of the context of information systems development -within an overall manufacturing systems engineering approach. A resulting methodology that addresses both information systems and physical-related issues would be of considerable benefit in providing manufacturing systems engineers with a disciplined and professional approach.

A second research topic concerns the implementation of software engineering and CASE within a manufacturing organisation. A defined plan for introducing both methods and tools, and facilitating the transfer of knowledge and guidance to manufacturing systems engineers, would help the successful application of CASE to manufacturing information Systems.

The main research proposal that builds on the results of this research topic is aimed at the identification of current practice for the specification of manufacturing information Systems within the UK manufacturing industry, the approaches used and the requirements of the manufacturing Systems engineer. From this, the main problems facing manufacturing information systems development can be identified. This will lead to the development of detailed requirements for the improved application of software engineering methods and CASE tools. A prototype CASE tool designed for specifying the requirements of manufacturing information systems will be developed. based on Execrator and with support from the vendor, INTERSOLV. The aim is to identify what is required by the manufacturing Systems engineer and the supporting CASE tools, by the identification of current practice for the specification of information Systems within UK manufacturing organisations. This will lead to the development of detailed requirements for the effective use of software engineering methodologies and front-end CASE tools in supporting the typical manufacturing systems engineer. A prototype CASE product will be constructed to fulfil these requirements and Is seen as the basis for considerable commercial exploration.

CONCLUSION

This paper presents the research undertaken in two traditionally separate areas, manufacturing systems engineering and information systems development The research has reinforced the need for structured approaches and tools in enabling manufacturing Systems engineers to analyse and specify information systems in support of manufacturing.

The relevance of software engineering methodologies and supporting CASE tools has been established in a more detailed manner than has previously been under-taken. The main developments identified in this work relating to the effective adoption of CASE within the manufacturing industry have been presented. These developments relate to the orientation of the methods towards manufacturing systems engineers, the ease of use of the supporting CASE tools, and the planned introduction of the methods and tools. Substantial further research has been presented, which Is aimed at extending the research work by defining the detailed requirements of manufacturing Systems engineers and appropriate CASE methods and tools In specifying manufacturing information Systems.

Acknowledgments

The research work was undertaken with the support of the ACME Directorate of the SERC under the CASE award scheme.

The authors would like to thank the collaborating organisations, and the experts from academia and industry who found time to contribute to this work

CLIENT-CENTRIC CASE

Since the early 1980s, AT&T ISTEL has been operating a business and software process re-engineering programme. The paper presents the instrumental role of the AT&T ISTEL in-house corporate CASE in providing holistic system generation and class leading productivity levels. The paper focuses on how the evolving needs of different classes of client - both internal and external - have been met. The architecture, operation and achievements of both the production and development CASE toolsets are summarised. A brief assessment of the AT&T ISTEL current CASE position is also given, followed ~ an extrapolation to the position in the year 2000.

Introduction

For nearly a decade, AT&T ISTEL (ATTI) has been pursuing a process **improvement programme** involving CASE. As this programme is business critical and its goals timeless, a critical success factor for the CASE tool involved has been its ability to continuously improve as the business processes it supports need to improve.

Ultimately, these business processes serve external clients by providing operational software of value to their businesses. These businesses are in a variety of vertical markets4 but are all characterised by Increasing corn-petition and rate of change. These characteristics are reflected across the supply boundary, placing inexorable pressure on delivery processes to deliver and evolve. In turn, they place pressure on any CASE-based support processes to deliver and evolve. Genuinely client-centric CASE is thus a critical business requirement for AT&T ISTEL.

To this end, we have internally developed and deployed a production CASE tool which itself has been engineered using an internal development CASE tool. In this paper, we demonstrate in both qualitative and quantitative terms the pivotal role of this CASE engineered CASE in optimising key software processes within our organisation. Although the locus of the paper is on CASE, and consequentially its tone is technological, the overall point being made is a strong corn mercurial one; in order to provide clients with consistent quality products, you need consistent business processes; to provide such products competitively, you need to enable and then control process change, and if CASE has a pivotal role in that process evolution, it itself must be capable of controlled optimisation.

A measure of the success of the approach presented in this paper is that the core ATTI business process of product development is now five to ten times more productive than a year ago, and with increased quality. This is the conclusion of an Internal client' developing a multi-million pound commercial product.

Throughout this paper, three classes of client are encountered; clients external to ATTI who procure our software; clients internal to ATTI who utilise the production CASE toolset to develop the software; and the developers of both CASE toolsets who, as they use both to develop operational systems, are their own clients. The title of this paper

reflects the need to reconcile and satisfy the needs of all three in order to compete and prosper.

Background

In 1985, a small development team was set up to improve ISTEL's core system development processes. The central vision of the team was to eliminate manual programming by establishing rigorous, logical definitions of business solutions which could be. automatically translated into an executable form. Evaluation of tools and methods at that time had led to the adoption of Jackson Systems Development (JSD) [1] and Jackson Structured Programming (JSD) [2], together with proprietary tools.

In 1987, the ISTEL quality programme adopted Crosby's quality philosophy and embodied it in company standards and working practices. Over the next few, years, dynamic changes to the software development process took place as manual practices were replaced by progressive code generation. However, by the time the company became part of AT&T in 1989, this process re-engineering initiative was becoming Increasingly constrained by the Inability of third-party tools to evolve alongside the house generators. This led to a revaluation of the tool and method support processes and the consequent founding of the ISTEL Applications Architecture (IAA) programme.

The IAA stated goal is 'to produce and deliver quality, high specification, portable software (VADS, product, bespoke) using less human resource and in a shorter timeframe than our competitors'. This goal is effectively a mission statement for the IM programme. The context within which the original goal was framed meant that the software referred to in the statement was end-user, i.e. for use by external clients. However, over the years, the IAA team has interpreted it in a wider sense encompassing software delivered to internal clients and software developed for IAA use only. Until recently, however, the scope of the IAA lifecycle has been constant, with requirements capture and validation outside the IAA CASE boundary.

In the following, we aim to show how and why an in-house CASE toolset has been developed by the IAA team and to do so within the current business process context. We also present the findings of its intense use by an internal client, the lessons learnt and speculate briefly on future prospects.

IAA processes and toolset

Fig. 25 presents an overview of the business processes with which IAA Is directly Involved. They are a set of dependent processes whose external interface is with the client; taking their needs as input and providing business solutions as output. The scope of these business solutions covers the AT&T ISTEL major applications domains; value added and data services (ADS) and commercial software products into focused vertical markets (e.g. manufacturing). The goal of IAA was therefore Initially focused on solving problems in these domains, although the approach taken has created the potential for much wider applicability.

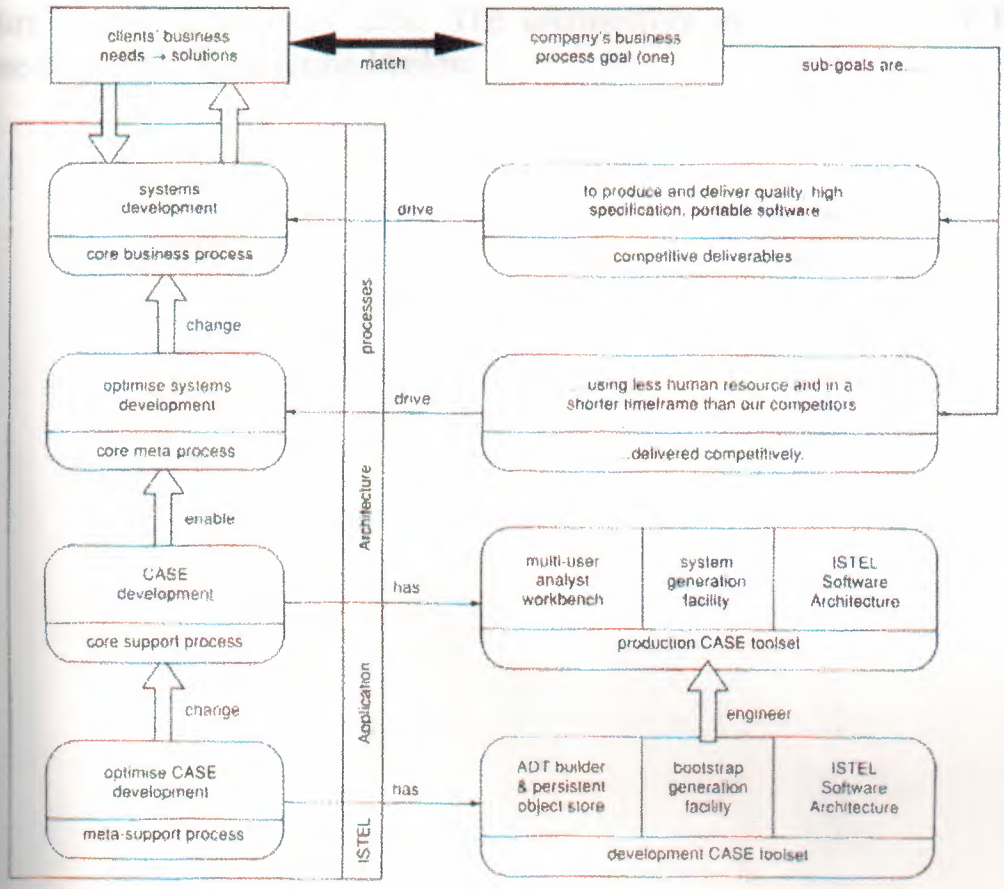


Figure 22. IAA processes and production/development CASE toolset.

The core business process shown in fig. 22, Systems development, is driven by the first Part of the RA goal to produce competitive deliverable's. The quality requirement is met by the continued adherence to Crosby's Principles of Quality Management and the choice of

development methodology (an IM variant of JSD), where conformance to specification can be guaranteed via automate transformations. The high specification requirement is also met by JSD, which Is a rich language covering the three dimensions of systems development (data. function and time) In a concise and rigorous manner. It also facilitates client discussion of the business model by capturing the business entities and their behaviour in a statically reviewable form.

The portability requirement is met by preserving the separation of logical and physical concerns provided by JSD when implementing solutions within the ISTEL. Software Architecture (ISA). Basically, ISA provides a reliable execution environment where the look-and-feel of the business application is cleanly separated from Its business logic, and that In turn from the business data. The architecture and operation of this "three-box-trick" is described below.

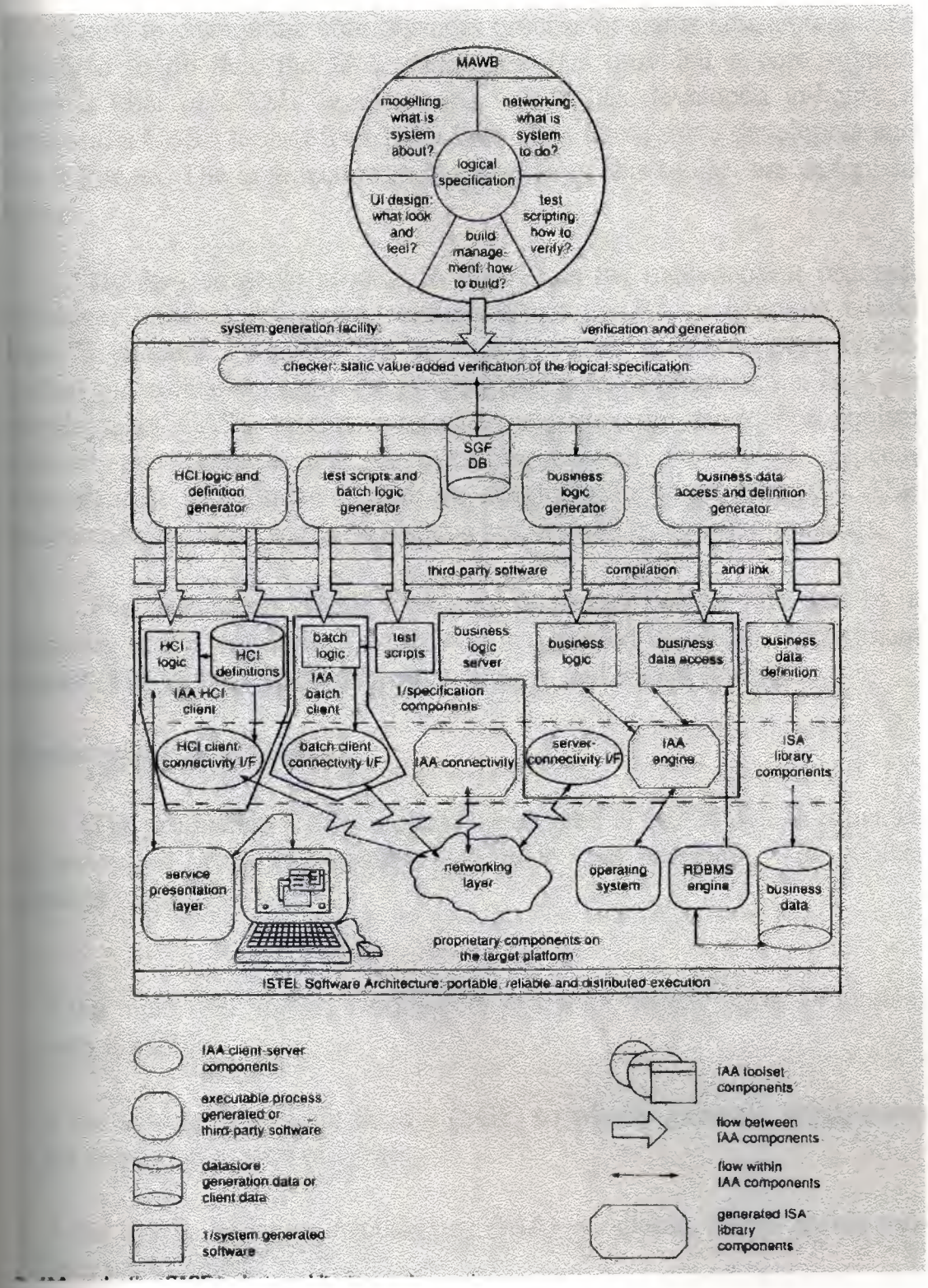


Figure 23. IAA production CASE toolset; architecture and operation.

The core meta-process shown in Fig. 23, optimise Systems development, is driven by the second part of the IM goal to produce deliverable's commonly. This RA process Is a mete-process, In that its

function is to change the core business process of stems development. To enable it to do this, the IAA programme has provided a core support process (that of CASE development), which has developed in-house a production CASE toolset for use by both front-line system developers and the RA team. The architecture and operation of this toolset are described below.

The key strategic process in Fig. 23 is the meta-support process, optimise CASE development, without which the RA programme would have floundered by now. The process provides a *development* CASE toolset for exclusive use by the production CASE developers, i.e. IAA the development CASE toolset is used to engineer the production toolset without this process and its attendant toolset, the ISTEL process improvement programme would be constrained by its internal CASE products in the same way as it became constrained by external products.

The production CASE toolset Is presented below, followed by a summary of the way ft itself was CASE engineered using the development toolset. The toolsets share many key features and design principles.

Production CASE toolset

The production CASE toolset is presented in fig. 24, which provides an overview of its components and their inter-operation. The toolset consists of

- the multi-user analyst workbench (MAWR), which works In tandem with the SOF.

- the system generation facility (SGF) to wholly generate applications for execution as part of the ISA

- the ISTEL Software Architecture (ISA), a reliable, client- server run-time environment.

The Systems development life-cycle under RA is thus an iterative life-cycle of

o *specify logically* what operational system is required in the MAWB.

o *verify and generate* it using the SGF to provide the target platform source, e.g. C and ESQL

o *build* that source into executable's using third-party compilers, pre-processors etc.

o *execute* the generated system within a controlled ISA environment

In operational terms, the system specifier sees a windows and objects-based workbench from which emerges either a complete running application ready for batch or Interactive testing, or a just of errors whose correction Is a prerequisite for generation. The theme is a variation on WISIWYG (what you specify (correctly) is what you get). In the following, we present the life-cycle in detail.

Logical specification

Specifications are captured in the MAWB (Fig. XXX) Using a commercially enhanced version of JSD; JSD modelling captures the business model, and then the networking phase builds business functionality around that model in an incremental fashion. The user interface (UI) for the system Is designed using the UI Design Tool (UIDT) embedded in the MAWR; the UIDT allows presentation, navigation and user access to be separately specified. Testing of the business logic is supported at the specification stage by the provision of a structured test script editor. Test scripts containing input data and expected output data can be constructed by linking directly to the relevant data definitions for the specification under test. Analyst-specified subsets of specifications are submitted by the build management to the SGF for generation.

Verification and generation

The SGF processes specifications exported from the MAWES (fig. 24), checks them thoroughly for consistency, completeness and reasonableness, and, if clean, proceeds to transform the logical specification into three main classes of output; HCI definitions and logic to create the user interface; procedural code to define the business logic,' and DML and DDL to describe the database access and structure. The SGP also processes exports of test scripts, and generates test harnesses capable

of exercising the business logic and automatically comparing expected outputs with actual outputs.

Currently, the 'back-end' of the SGF produces the three classes of output for an open systems environment:

business logic is C with embedded SQL for the database access.

schemas are SQL (ANSI conformant).

definitions are in a third-party GUE builder's format; the logic is mainly C, called by a layer of the GUI builder's 4GL

In the following, we describe the processing of this to form an executing application. It is Important to emphasise that the IAA team considers the execution environment to be within its CASE toolset boundary. The specification and generation stages are predicated on that environment having certain properties, some of which are provided by (generated) once-per-environment components.

Build and execution

Following generation, third-party software Is then employed to build (i.e. compile and link) the generated output Into a configuration fit for execution (Fig. 25). This configuration Is basically a client-server configuration, with generated HCI or batches clients having their requests exclusively brokenness by IAA Connectively, a distribution management system.

The clients, servers and connective itself run as distinct operating system processes and can be located anywhere on the network of available processors. The client processes bind In components generated for the end-user application with ISA library components to form a unit capable of communicating with the connectivity processes. HCI clients also bind in libraries from the third-party GUI builder, which then Interface at run-time with the GUI builder's service presentation layer to give the windowed HCI. Batch clients interface with users via the operating system command line.

Connectivity Is Itself an IAA specified and generated program with its own business logic and database; the presentation in fig. 24 Is relatively

simple at this point Connectivity then communicates through a server ISA library component to the business logic for the application concerned. To function completely and correctly, the business logic server process also includes the database access code and the IAA engine.

The IAA engine (fig. 25) is basically a method-smart transaction monitor, which mediates between the NC!, the business logic and business data. It guarantees transaction integrity and improves the portability' of generated output by insulating it from the target platform; the IAA JSD variant has transaction extensions to allow analysts to specify transactions (committable units of work) unambiguously and reliably.

The IAA engine provides *dynamic* verification of the executing application to complement the *static* verification provided by the checker component of the SOF (Fig. 26). Basically, the checker knows about 'JSD at rest'; the engine knows about it 'in motion' Both work in specifi*cation terms,* not that of the target platform (e.g. C)- For example, when the engine provides a trace of a successful transaction, or a dump of an aborted transaction, ft does so in terms of JSD. Thus, specifiers can work in specification terms throughout the entire life-cycle.

Furthermore, as for IAA connectivity, the engine is itself an IAA generated system. The SOP is capable of generating it either as a 'bottomed-out' component (i.e. it can run without a need for another 'engine' below it) or as if it were an end-user application. The latter is done in order to automatically batch regression test it by running it on top of a bottomed-out version of itself.

Let us return briefly to client needs; a key lesson that the IAA team has learnt here, 'which we believe is portable, is that CASE tools used for developing high-Integrity systems must have both static and dynamic verification capabilities, as exemplified 'within IAA by the checker and engine described above.

Engineering the production CASE toolset

Fig. 26 shows the production toolset being 'engineered' using the development CASE toolset. Fig. 27 expands on this process.

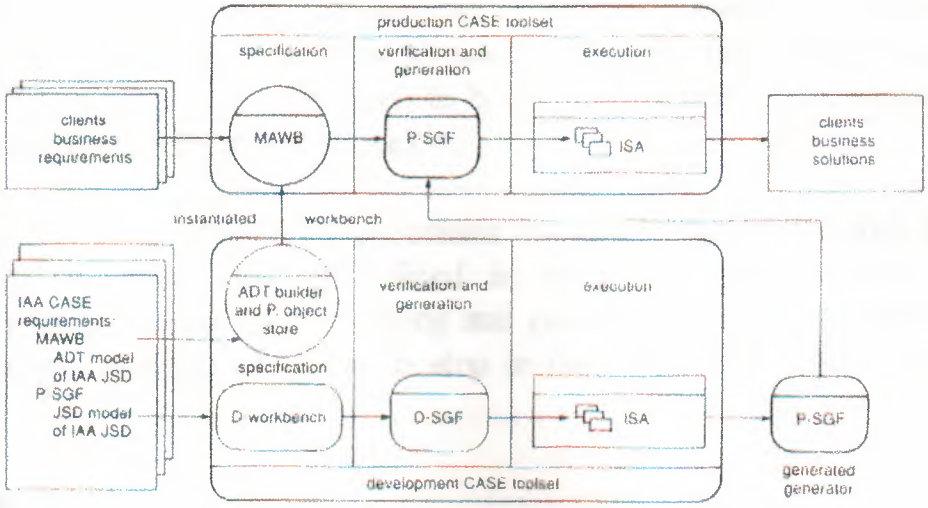


Figure 24. CASE engineering the production CASE toolset.

In the case of the MAWB, it is engineered using an abstract data type (ADT) building tool, where the MAWR structure and operation is basically an instantiation of a complex ADT (>800 sul>ADTs). Users' data are then instances of those ADTs. Each ADT has a set of associated views and operations, which can be used to design the manner in which users' specifications are viewed and manipulated.

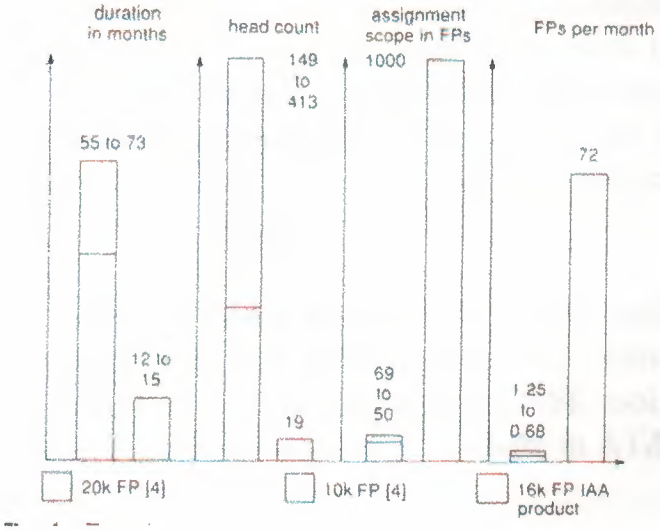


Figure 25. Function point evaluation of the production CASE toolset.

In the case of the production SOF *(P'SGF,,* this Is engineered by specifying the P-SGF in JSQ and then using a manually built development

SGF (ID-SOF) to check and generate it This is preferable to manually writing the 2 million + lines of C which compose the P-SGF business logic. The same process was used to build the production RA engine and RA connectivity (described above).

Both these engineering processes are fundamentally 'closed', in the MAWB, the root ADT is defined in terms of itself; P-SGF, once bootstrapped, is capable of checking and generating itself. This solves the fundamental problem of where to stop in the chain of CASE supporting CASE.

Present status

The production toolset has been in constant use with an internal client since late 1992 and has evolved via a process of evolutionary delivery as advocated by Cub [3]. The internal client has been developing a multi-million pound commercial product for their clients In a key vertical market.

Throughout 1993, hundreds of internal client change requests were raised and embodied in internal releases of the toolset (over 20). Each major release has represented a significant increase in both functionality and performance, e.g. P-SGF generation time shrinks by a factor of seven, the MAWR data repository is halved in size etc. This evolutionary delivery process has been made possible by having what Gub refers to as an open architecture; we use the term flexible architecture. Both functionality and performance (non-functional attributes In Glib's terms) can be refined in the MAWS, SOF and ISA areas In a controlled yet responsive manner because of their flexible architectures.

In quantitative terms, the IAA team has delivered results. Recently, an internal evaluation of the IAA performance was commissioned at executive level with regard to its productivity as a CASE tool. To facilitate potential comparison with other tools in use elsewhere in AT&T, Jones' [4] function point analysis was applied.

In brief, the size of product developed by the IAA Internal client places it between the 10k and 20k function point thresholds table [4]. This table shows function paint scaled metrics for software projects In Jones' database and gives typical US industry figures; four metrics were of interest the Internal evaluation, project duration, staffing level, assignment scope and production rate. Fig. 25 presents these for the 10k and 20k

thresholds and compares them with those achieved by the use of IAA on a multi-million pound commercial product.

Obviously, there are assumptions In mapping the IAA specifications to function points. The figures given for IAA fail in the middle of a range of figures obtained by varying some of the key assumptions. Jones' figures have been Informally but independently validated by comparison with metrics for equivalent products being developed by our competitors using traditional techniques; their performance is in the range of Jones[1] figures, not the IAA figures (e.g. project duration of many years, not months, hundreds of development staff, not dozens, etc).

In addition to this internal productivity study, IAA has also been the subject of a recent evaluation of CASE and methodologies in general, which reconsidered the company's original aims concerning the efficiency and effectiveness of its Systems development processes. Although Internal, it was an independent and thorough study, and it set IAA in a fresh perspective by evaluating it alongside the current offerings in the CASE and method arena.

The scope of the evaluation was application product development from specification through to delivery, and the primary requirements were responsiveness, flexibility, portability and Inter-operability. Of about 30 initial contenders, four were short-listed for detailed evaluation; IAA was one, there were two CASE/40L combinations based on conventional structured methods, and an object-oriented CASE/4GL combination.

The key observation arising from the evaluation was that the current weakness of the IAA (highly interactive prototyping and to a lesser extent portability and inter-operability) was the CASE/4GL candidates' strength. However, for the evaluation of time4omarket and quality-with-productivity, the situation was reversed. Overall, IAA was assessed as having the most attractive Investrnent benefit/risk profile, and it became the recommended approach for company product development

Future issues

Looking to the year 2000, the IAA strategic view of CASE continues to be one of ensuring that it is a client 'slave', and not a

technologically mesmerising 'master'. The IAA toolset has an architecture capable of extension in many client serving directions.

Our current intention is to extend the IAA CASE bound. any to encompass more of the early lifecycle, provide richer execution at the end of the lifecycle, and augment the development support in between.

In order to encompass more of the early lifecycle, we are currently assessing methods for requirements capture and validation to dovetail with the middle-out modelling nature of JSD. To provide a richer execution, we have recently been generating Systems to work with multi-media HCI's, and assessing the methodological and technical Issues arising from richer presentation options. To augment the development support, we are tackling criticisms about prototyping capability by investigating the merits of interpretive specification animation; this would provide an informal but indicative preview of the specification, prior to rigorous verification and subsequent generation of the final System.

All of these initiatives share one fundamental criterion; how will a particular modification to the toolset ultimately add value to a client's business? Successful evolution of the IAA CASE toolset will depend on dear and consistent application of this client-centric criterion.

Acknowledgments

The author would like to thank John Alexander, Dave Bin keman, Ken Hicks, John Hardy-, Martin Kendall, Paul King, Mark and Mike Newman, James Petry, Rob Silk Jon Twigge for their work on IAA; and the ATTI directors for their perennial support.

CASE IN THE THIRD GENERATION

The paper outlines and analyses the development of CASE and proposes a direction for research and development. The focus is on analyst workbenches, rather than other types of CASE tool, but the proposals and conclusions are general applicable to tools supporting other phases of the development life-cycle. The current generation of CASE tools has no knowledge of the software process. An architecture that combines a management control superstructure with the normal development tools Is proposed for CASE in the next generation. The design of a superstructure, based on an existing CASE tool, is discussed.

Introduction

In the mid- I 970s, structured methods of Systems analysis and design were proposed [1, 2]. These were based on accessible graphical notations, principally the data flow diagram (DFD), and gave rise to computer-aided software engineering (CASF) tools. Rock-Evans [3] defined CASE tools as *'software* packages *which automate or support one or more of the activities of the systems life* cycle'.

Although this is a good definition, it should also be said that in the early days of CASE, the term was synonymous 'with support tools for Systems analysis and design. Later, the terms 'upper' and 'lower' CASE were used to denote tools at the start and end of the systems life-cycle [41. Others categorised tools as either upper, middle or lower CASE [5]. In order to avoid confusion, the term analyst workbench ~WB) is used in this paper to denote CASE tools that support systems analysis and design. Rock-Evans [3] defined an AWB as 'an *Interactive software product which aids or automates some or all* of *the* tasks *of business requirements analysis'*

Norman and Chen [6] consider two generations of CASE tools (AWBs). The first generation appeared at the time of the advent of structured methods. This was an anempt to automate structured methods using the available technology, i.e. the tools were text-based and ran on a mainframe computer. An example of this type of CASE tool Is PSL/PSA [7] when microcomputers and work-stations 'with graphical user interfaces (GUIs) were introduced in the 1 980s, the second generation of AWESs appeared. These supported the graphical notations of structured methods. There are many of these tools on the market, e.g. Excelerator, IEW, Teamwork, Systems Archited and IEF. Although these tools have been improved and enhanced considerably since they were first intr~ duced, there has been no further stepchange in the provision. Progress towards a third-generation AWS is the subject of this paper.

Over the last ten years, others have been working on a much grander vision of software support for the systems lifecycle; the Integrated Project Support Environment (IPSE). An IPSE was defined by Alvey [8] as

'a compatible set of specification, design, programming, building and testing tools supporting a development methodology which covers the entire life cycle, together with management control tools and procedures, all wing a coordinating and consistent project database.'

For reasons that are well documented [9, 10], IPSE's never progressed beyond the research laboratories. Indeed, as much of the research was intended to be precompetitive this might have been expected. In addition, many of the European and American projects are still ongoing, e.g. ALF [11], ESF [12], STARS and Arcadia [13].

However, the last five years have seen some of the more important research ideas from the IPSE projects filtering across to commercial CASE tools. In fact, in retrospect, one view would be that the IPSE projects acted as the pure research stream that generated ideas to be applied to CASE tools.

Some of the IPSE projects had a generic tool that allowed the development of point tools, e.g. ISTAR [l4] and ECLIPSE [15]. This gave rise to a class of tools known as meta-CASE or configurable CASE, and several appeared on the market, e.g. "'SF, Tool Builders Kit These tools are as yet at a very early stage. One stud"" has found that configurable CASE tools are difficult to configure and limited In what can be produced 'without resorting to coding in a third generation programming language [16]. It should also be noted that once a CASE tool Is configured via a meta-CASE tool, ft operates exactly like any other second generation CASE tool, i.e. configurable CASE tools represent only a potential short cut in the delivery. The tool itself represents no advance over others that were delivered via a different route.

Another important idea in IPSE research was that of integration. Given that most IPSEs wanted to use 'foreign' tools (i.e. tools -that had not been developed specially for the IPSE), there needed to be some way of integrating them; they needed to communicate and share data. This research gave rise to several CASE tools that might be described as integration frameworks. In tact, they are similar to IPSEs. Examples of these are HP Soft bench, DEC Cohesion and Atherton Software Back plane. These tools became known as Component CASE (or C-CASE) tools, presumably because components (i.e. tools) could be plugged in to the framework.

The use of C-CASE is also at a very early stage. A recent study [17] found that there are few examples of operational CASE tools being built using C-CASE (even though there is some interest in 'playing' with them). Currently, one of the main uses of such tools is to Integrate AWBs with documentation tools. Even this fairly unambitious integration involves a significant delivery time. This leads to the conclusion that unless

integration mechanisms improve dramatically, there Is no future for this technology. The view of Brown and Mcbermid [18] is that

'...the way to get integrated toolsets is to design and develop integrated toolsets, not to provide generic mechanisms and to hope that integration will arise as if by magic'.

Process modelling

On the subject of integration, Brown and McDermid [18] have defined a hierarchy of Integration between tools from method level down to carder level. The most desirable level of integration is method level integration, where tools are used, and interact, only 'within the context of the software development process.' This could be realised by having a process model that coordinated the use of the tools in the toolset This introduces the other important idea in IPSE research; process modelling. This is concerned with modelling the software development process. In simple terms, a process model consists of a network of project activities and the deliverable's or design artifacts that flow between them.

Process modelling is of interest to a number of different groups within computing. Much process modelling research has been under the auspices of or linked to IPSE research. Owing to this there has been a concentration on 'what has become known as process programming [19]; where the process model is actually encoded as a program and runs 'within the IPSE to control and guide the use of the tools in the environment Some would argue that there has been an over-concentration on process programming in process modelling research, at the expense of human considerations [20, 21]. However, apart from a few notable exceptions, it is fair to say that most researchers are enthusiastic about process programming. This is the application of process modelling that is pursued in this paper.

Most process modelling systems are embedded within IPSES in research projects. However, a number of tools have recently come on to the market as stand-alone process modellers [22]. The reason for this heightened interest in process modelling tools in the market-place is the current zeal for business process re-engineering (BPR) [23]. This concerns the re-engineering of organisational structures around business processes, rather than along traditional departmental lines. These tools might be used to model software processes, but they do not address process

programming; they are not intended to be used in conjunction 'with the tools in a CASE environment

Process modelling is also of Interest to the Quality community. An important concept for this group is the capability maturity model [24], which identifies five levels of process maturity. Here the focus is on modelling software processes in order to assess their maturity and improve them. Most industrial software processes are currently at the lower levels of maturity. There Is some debate as to whether a software process should be at a particular level before it is worth using CASE, or whether the use of CASE causes a process to be assessed and formalised, thus improving maturity.

The benefits of process modelling are well documented [25, 26]. The best chance that we have to make improvements in software productivity and quality lies with making improvements to the software process. This is not possible without software processes that are rigorously modelled and enacted to provide feedback for process improvement

Towards a third generation for CASE

The vision of the next generation of AWBs is one where the user does not have to know the software development process and invoke tools based on that knowledge. Rather, the primary Interaction 'with the AWI3 is with a process model that could guide the user in the steps of the process and control the use of the tools in the workbench. The second generation of AWEs is characterised by the support of the graphical notations of structured methods. The interaction here is directly with a tool in the work-bench via a GUI. The third generation should be characterised by the support of a process model integrated with the other tools in the workbench. The primary interaction here is not directly with any of the point tools, but with the process model.

If the way in which all of the techniques of a method fit together and relate to each other is considered, then a superstructure for an AWS might be defined. This would be concerned with process modelling to coordinate the lower level technique support facilities; AWBs could progress from being an uncoordinated collection of tools towards an environment that also defined how the tools should be used within the framework provided by a method. The use of tools at inappropriate points in the method could then be prohibited, and assistance could be offered to the user In the form of guidance from one stage of the method to the next.

This is not a new concept in general terms. In 1981 Wasserman [27] concluded that

'tools must therefore provide not only technical support for a specific phase of the life cycle, but should also provide management assistance and should facilitate the transition from one phase of *the life cycle* to *another (both forward and backward)'.*

Alvey, in the definition of an IPSIE, also saw the need for *management control tools'* as well as 'tools *supporting a development methodology'.*

However, this has patently not happened, particularly in the CASE arena. As Kaiser et at. observed about CASE tools [28], 'Most software tools are moronic assistants that know what to do but do not understand... how their tasks fit into the development process'. This is still true today, and the integration of CASE and process modelling has yet to be properly investigated.

The tight Integration of process modelling with development tools could lead to real method level integration. Rader *et* al. [17] found that there is a strong industrial awareness of the need to integrate process and tools. They concluded that the first step up the integration ladder from isolated CASE tools is to integrate clusters of tools, i.e. collections of tools supporting one part of the process. For this reason, It was decided to apply these ideas initially to an AWS.

CASE has long aspired to process modelling, even if it did not know its name. For example, the SSADM Conformance Appraisal Scheme [29] devised a system for star-rating AWBs supporting SSADM. The highest rating was to be given to a tool that *'directs SSADM stages and steps'.* McClure [30] also argued that software productivity would only be improved by CASE took if they acted as methodology companions. Vessey [31] define methodology companions as a tool that *'facilitates following me steps and rules of a structured method'.* This is analogous to process modelling. This view was also supported by Hatley [32].

It is apparent that the superstructure of a third-generation AWB (3GAWB) might encompass not only process modelling, but also other management control activities. For example, in the Aspect Project 1331, the process model is called the 'project plan'. In the PMMS Project [34],

the process model was seen as an automated quality plan. This indicated that process modelling was intertwined with project and quality management, and that all three needed to be considered together in order to arrive at any sensible conclusions, The view' that these three activities need to be considered alongside technical development In any support software Is endorsed by many researchers [35, 36].

3GAWB architecture

The first step towards realising the vision of a 3GAWB was to define a general architecture. The initial architecture defined is shown in Fig. 26.

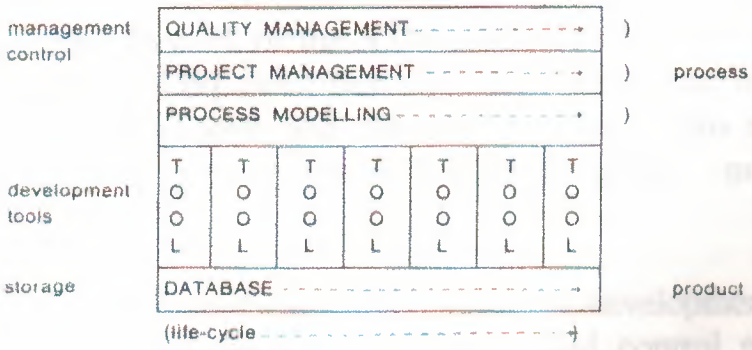


Figure 26. Initial 3GAWB architecture.

At the centre of the model there are a number of development or point tools. In a Yourdon AWB, for example, these would be a DFID editor, an Entity Relationship Diagram editor, a data dictionary a Structure Charter etc. It is useful to consider these as having application at particular points in the lifecycle. Thus, the tools towards the left of fig.26 would be the early life-cycle tools. Obviously, no set or tools for a method would follow such a simplistic pattern; they might be used in parallel or repeatedly throughout the life-cycle (this is pursued later). The tools would store data and communicate through an underlying database. This is the current state of commercial CASE tools or 2GAWBs.

In addition to a substructure concerned with the products of systems development there is also a superstructure concerned 'with the process of systems development The three strands of the superstructure could be based around a process model, a project plan and a quality plan,

respectively. All three parts of this superstructure apply throughout the life-cycle, rather than at a particular point.

Similar diagrams to this have also been used by Steinbauer [37] and Bjorner and Prehn [38]. They seem to be the embodiment of the Alvey IPSE vision. This identifies three types of tools; 'tools supporting a development methodology', 'management control tools' and a 'project database'. These are evident as the centre, superstructure and substructure of these diagrams, respectively.

There is some direct support for the components of the superstructure of the 3GAWB architecture proposed above. For example, Fernstrom and Ohisson [36] discuss process enacted environments and show the coincidence of process modelling, project management and quality management. In work on the ATMOSPHERE Systems Engineering Environment, Obbink [39] describes tools to support the management process, the life-cycle process and the quality process. This maps on to project management, process modelling and quality management, respectively.

Chroust [35] defines three 'paths' called the development path, the quality control path and the project planning and control path. These correspond to process modelling, quality management and project management The distinction that Chroust makes between quality management and quality control is important, however. Quality management comprises quality assurance and quality control [40]. It is clear that many quality assurance activities could not be automated within an AWES. For example, reviewing the quality management system itself, or checking that the common functions of an organisation, such as purchasing, are operating efficiently are necessarily human tasks. Quality control, on the other hand, operates at project level to define methods, standards and checks for the project There is obviously' some scope here for automation within an AWS. Therefore, the name of the quality management strand of the superstructure Is changed to 'quality control'.

Another point of discussion concerning the quality strand relates to configuration management (which is part of quality management). Several researchers have drawn this out as a separate entity. Dowson [41] argues that the software manufacturing process should address

'technical development, project management, data and configuration management, and quality assurance and control throughout the software lifecycle'.

Gofiner [42] says that one requirement for an IPSE Is support for development, project management, quality assurance, configuration management and the life-cycle.

In order that the 3GAWB architecture is as precise as possible, configuration management is extracted from the quality strand and shown explicitly. This also makes sense If the process and product are to be abstracted. Configuration management clearly concerns the product Therefore, it should be removed from the quality strand of the super-structure and added to the sub-structure.

This gives rise to a revised model of the architecture of a 3GAWB, as shown in Fig. 29. This could be viewed as a generic model of a 3GAWB architecture because it is not particular to any one method. It could be instantiated for a specific method, however. The change to the model could be to name specific development tools and put them in the appropriate places in the life-cycle. For example, an instantiated 3GAWB architecture for the Yourdon method could be drawn as in Fig. 27.

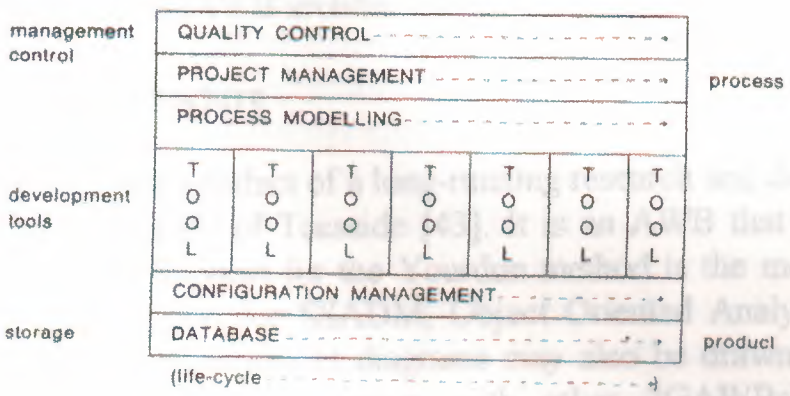


Figure 27. Generic 3GAWB architecture.

Notice how the sequence and parallelism of tool use throughout the life-cycle may be shown for a specific method. Notice also the inclusion or a word processor as one of the development tools. At first sight, this seems odd. When thinking of development tools, examples like DFD editors and

ERD editors come readily to mind. It is obvious that 'word processors are used for preparing reports at certain points In a method. However, in most methods, word processors could also be used for some of the 'technical' tasks. For example, in Yourdon, they could be used for things like drawing up event lists and process specifications. Therefore1 the provision of or easy access to a word processor is an important general requirement for the development tools of an AWB.

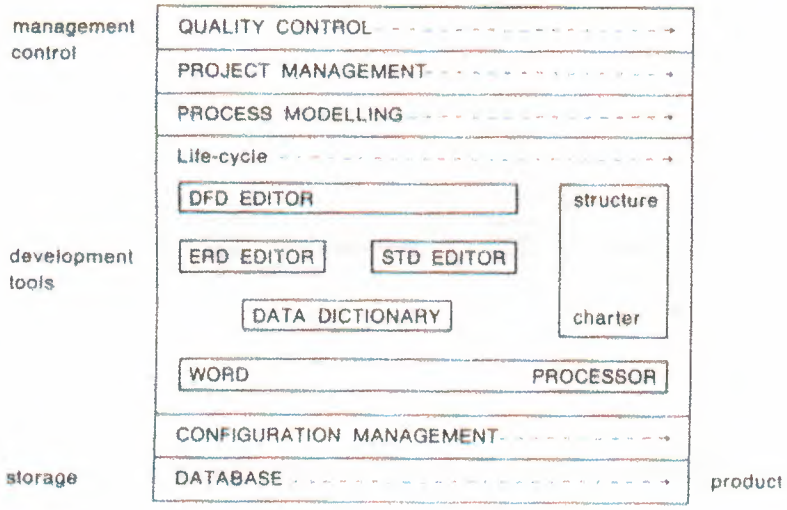


Figure 28. Yourdon 3GAWB architecture.

Design of a superstructure

ASCENT is the product of a long-running research and development project at the University of Teesside [43]. It is an AWB that supports a variety of methods. Support for the Yourdon method is the most mature, but there is also support for SSADM, Object Oriented Analysis (OOA) and Mascot Project management diagrams may also be drawn. Referring to Fig. 29. ASCENT (in common with other 2GAWBs) provides development tools and database storage.

In order to bring ASCENT Into the third generation, a management control superstructure must be developed. The requirements of a superstructure involving process modelling, project management and quality control have now been defined. The design of the superstructure has been started, and the processing behind the process modelling strand has been specified. Some of the process modelling screens have also been designed.

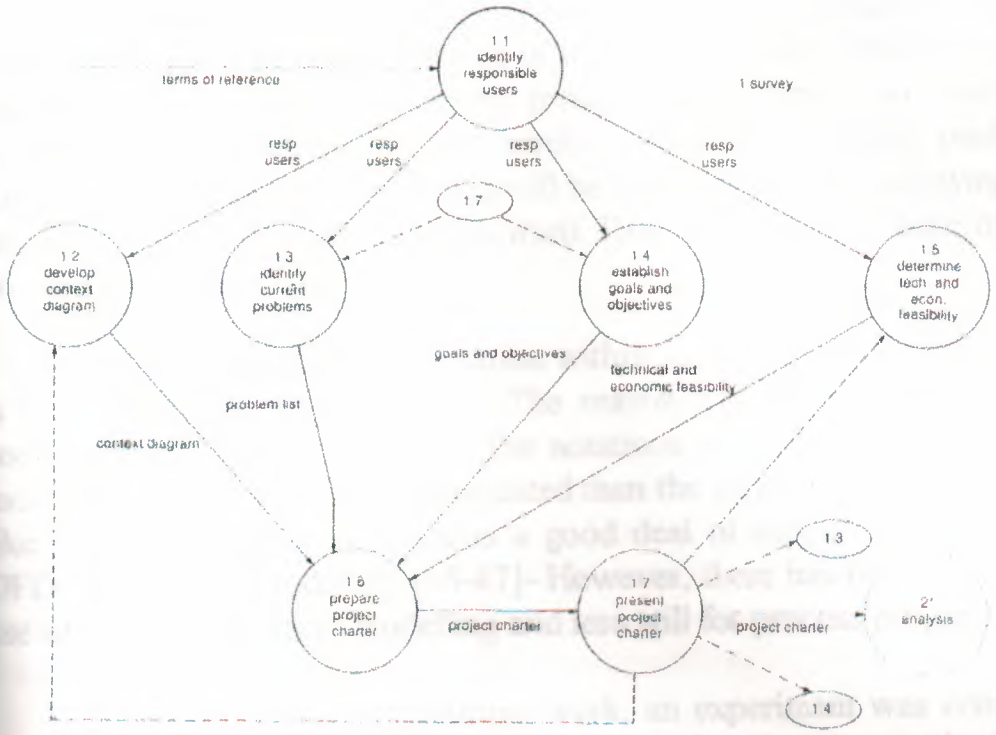


Figure 29. Example DFD from the Yourdon process model.

The process modelling strand of the superstructure will provide the primary interface for both developers and managers. When a developer logs on to a project a process list is presented that shows the atomic processes assigned to the developer in the project This will also show their states and whether they are runnable. Runnable processes may be selected, and a list of tools for the process appears. Selection of one of these will transfer control to the tool. This requires the development tools in the environment (e.g. DFD and ERD editor) to be logically separate. Only data indicated by the process model as being pertinent to the process (i.e. its inputs) may be accessed by the tool, sometimes in read only mode. when a process is completed, or iteration Is identified, states and runnable flags will normally change, thus updating the 'live' processes.

For managers, the process modelling strand will provide facilities to define process models. These could be developed afresh or customised from a library of process models. It is important that actual process models are available, as well as mechanisms to define them. Facilities are also provided to instantiated generic process models for particular projects.

Both developers and managers will be alerted to project events and be able to Inspect the current status of the process model while it is being enacted. Project events, such as processes completing or becoming runnable, will be transmitted in a similar fashion to electronic mail. The current status of the process model will be communicated by allowing read access to the actual process model itself This 'will show the state of each process graphically'.

The process modelling notation within the process modelling strand is based on data flow diagrams. The reason for this is a view of the confident with that of ince [44]; 'the notations used for software process modelling will be no more sophisticated than the simpler CASE nota dons, like data flow diagrams'. There is a good deal of support for the use of DFDs for process modelling [45-47]- However, there has been little actual use of DFDs for process modelling and less still for process programming.

Before undertaking the design work, an experiment was conducted with the construction of a process model for the Yourdon method using DFDs. Although It is beyond the scope of this paper to consider this In detail, the following gives some examples and outlines the main conclusions.

An early conclusion was that the basic structures thought to be necessary for process modelling (sequence, selection, iteration and refinement) could be supported. External entities and data stores were unnecessary because the Important part of the network was the processes and the way that data flowed between them to define precedence.

Concerning data flows, It was concluded that some of the more abstract flows sometimes shown on a standard DFD were superfluous. All that should be shown were design artifacts that passed between processes. An example from the DFD set for the Yourdon process model is presented in Fig. XXX. Notice the extension to the normal DFD notation to cope with iteration. Potential iteration from process 1.7 is shown via dashed arrows going back in the network or via connectors.

It was also concluded that the AND/OR ambiguity inherent in DFDs must be eradicated if the model was to be executable. This was done by importing the semantics often found in other graphical notations such as project management diagrams or Petri nets; everything is joined by an AND. The evolving Yourdon process model showed that this was the right

logical operator to assume in the vast majority of cases. In fact, an OR was only needed when part of the network was optional.

There were other reasons why it would be useful to identify optionality (i.e. instantiation), and it was concluded that the DFD notation should be extended to identity optional flows and processes. These were shown as dashed objects to intuitively convey their optional existence. Connection rules were also established such that optional processes could only receive or generate optional flows. Fig. 30 shows an optional process (2.1) which generates an optional flow.

There was some Information that needed to be recorded that was difficult to capture graphically, e.g. states, tools and personnel for processes. These were dealt with in a similar way to a standard data dictionary. Given these enhancements, structured analysis techniques were found to be eminently suitable for process modelling.

The automation of project management is mature, and it was quite straightforward to design the way that this would operate within the project management strand. The role of the process modelling strand was pivotal here, and the design of the project management strand was Incremental. This might have been expected because close similarities between the activities and supporting notations have been noted previously [48-50]. In fact, the conclusions here are that process modelling and project management should be provided via the same user interface, based on the same task model.

The provision of two separate user interfaces and two separate strands might be more attractive it they were sup-porting two recognised roles in a development team. Only one or these roles is currently recognised; that of the project manager. However, the benefits of process modelling could be delivered as an extension and automation or the project management role. An examination of the task models of process modelling and project management show that there are some differences, but many similarities. Further the extra process modelling information would be appropriate and useful to the project manager.

As project management is sited in an integrated 3GAWB, together with development and process modelling tools, the activity may be further enhanced. Integration with the development tools allows support for task identification and estimation. Assistance with task identification would not be automatic, but the design models produced by the development tools do

provide information about processes to be carried out later in the life-cycle. Assistance with estimation would not be at the level of individual processes, but design metrics could be produced to check on overall system size. Integration with the process modelling tools provides an enriched project plan that supports levelling and identifies iteration and optionality. It also provides an automated facility to control and guide the actual process of development, and receive Information about progress and pertinent project events.

Figure 30. Example DFD showing optionality.

This general conclusion also followed from the design work on the quality control strand. There has been little automation of quality control to date, except the word processing of the quality plan. Indeed, there is little opportunity for this as a stand-alone activity. However, there are automation opportunities for quality control as part of an environment that also includes the automation of development, process modelling and project management In fact, several of the requirements of quality control are met because process modelling and project management are assured.

The main additional functionality required for quality control relates to the keeping of quality records resulting from the design process. These could relate to anything of interest to quality control, e.g. reviews and

inspections, process and product measurements. Much of the supporting information and processes for these occurs naturally, but there Is some extra effort involved in producing the actual quality records. A general feature of quality records is their interest in the history of activities or measurements. All quality records should be held in the process model, open to inspection. These could be available as part of the dictionary.

Like project management, there is no physically separate strand of the AWS superstructure envisaged here, rather an augmentation of the existing functionality for quality control activities. Quality control is the responsibility of the project manager, and so the functionality for both activities should be available together.

Future work

The next step is the completion of the design of the superstructure and its implementation based on ASCENT. This will provide the crucial test of the feasibility of this approach. Once a prototype exists, it is important that it is used by developers and managers to provide feedback This should be sought at two levels. At a high level, it is important to determine if the approach is useful. At a lower level, it is important to obtain detailed feedback that would allow changes to be made to the workbench.

If the approach is valid, investigation might be carded out into the separation of the superstructure from ASCENT; it might be possible to implement the super-structure as a stand-alone management control tool that communicated with an AWBs via a generic integration mechanism. Although this is quite attractive, particularly commercially, it is doubtful that current Integration mechanisms could support this effectively.

A more viable proposition might be to look at applying these ideas to tools supporting other phases of the development life-cycle. It is clear that these ideas might be applied to other clusters of tools. For example, it seems highly likely that a similar management control superstructure could be imposed on a programmers workbench in a similar fashion. This phase lacks the strong process model provided by structured methods at the analysis phase, but the management control requirements are unchanged.

The logical progression is to integrate these clusters of tools into a single environment under a single management control superstructure. Processes on design models early in the life-cycle might usefully feed-

forward to become items of work later in the life-cycle. This might even provide the framework into which all CASE tools could be integrated,

Conclusions

CASE tools should be enhanced so that full life-cycle activities are integrated with development activities. This points the way towards an architecture for CASE in the third generation.

In this paper, an architecture has been proposed for a 3GAWB where development tools are augmented by a super structure of management control tools. Development tools should include editors for the techniques of structured methods and a word processor.

The main distinguishing characteristic of AWBs in the third generation is the primary interaction of users with the process model, rather than the development tools. The process model is used to guide developers in the steps of the method and control the use of the development tools in the environment. Process modelling should be based on accessible graphical techniques, and an enhanced data flow diagram/data dictionary notation is eminently suitable.

Another important characteristic of third generation tools is that they will be used not only by developers, but also by managers. Support should be provided for process modelling, project management and quality control. When these activities are supported In one environment, together with developmental activities, there is a cross-fertilisation of information that offers wider opportunities for automation. Although there are three distinct activities in the super-structure, or three logical strands, they should be implemented as a single physical strand (with a single user interface). Much of the functionality of this strand would be based on a single, unified task model. The user of this strand would be the project manager. This role is already responsible for project management and quality control, and would also carry out process modelling.

Early research results suggest that Implementation of the proposed architecture is achievable. It is particularly important that the superstructure could be added incremental to a second-generation CASE tool. This allows existing tools to migrate from the second to the third generation.

In the future, these ideas could be applied to other clusters of CASE tools relating to other phases of the development lifecycle. This could progress towards the integration of all types of CASE tools within a framework provided by a consistent and coherent management control superstructure, This vision of the future offers the best practical opportunity for realising the dream of an integrated project support environment.

Currently, tools in a CASE environment are invoked by developers based on process models that are in their heads, or at best in methodology or standards manuals. Within a third generation CASE environment the inter-action is directly with an explicit process model that provides guidance and control, in addition, the integrated automation of management control activities is provided. This is imperative for coherent management of software projects. Such a CASE environment will be instrumental in the push for process improvement and improvements in software productivity and quality, towards the year 2000.

KNOWLEDGE-BASED CASE TOOLS: IMPROVING PERFORMANCE USING DOMAIN-SPECIFIC KNOWLEDGE

Knowledge-based CASE tools are able to play an active part in the design of computer-based systems. Providing such tools with in-built domain-specific (or 'real world') knowledge can enhance both the performance and the appearance of Intelligence. However, little work has so far been carried out as to how this might be achieved. The paper Illustrates how such knowledge may be provided In the form of generic models based on a thesaurus approach, and applies the technique to a knowledge-based CASE tool designed to support object-oriented design.

Using domain-specific knowledge

The use of predefined domain-specific knowledge within a KB-CASE tool, and the ability to reason with and make use of this knowledge during a design session, can enhance the appearance of intelligence and increase the efficiency of a tool. The concept of providing domain-specific know-ledge for use by KBCASE tools exploits the fad that a high proportion of application systems are similar in nature (i.e are variations on recurring themes). For example, many systems designed in commercial environments support broadly similar functions across organisations, such as stock control, sales-order processing etc. Given this situation] the incorporation of domain-specific knowledge representing genetic models

of these common systems within a design tool, and the ability to reason with this knowledge, would be of obvious advantage. Such an approach can remove the need to ask trivial questions of the designer (thus enhancing the appearance of the intelligence of the tool) and may save time by removing the requirement to submit large amounts of initial domain knowledge.

Generic models may be used to exploit the similarity of such Systems by providing templates on which new systems may be based. Numerous possibilities exist for the exploitation of this approach. For example, the tool, having recognised an application domain, could present the generic model as an initial design attempt and customise it to the designer's requirements during the design session. Some work has been carried out in this area [15, 16]. However, the tools developed, although possessing domain knowledge in the form of internal dictionaries, could not present an initial design suggestion for the designer to consider. In general, very little 'work has taken place in this area to date [12, 17], despite the fact that the use of domain-specific knowledge can potentially yield numerous benefits. For example, domain-specific know-ledge can improve the overall performance of a KB-CASE tool in terms of the following:

o *Increased appearance of intelligence:* the tool appears to have previous knowledge of the application area. This apparent familiarity with the user's domain may in turn lead to greater acceptability.

o *Increased efficiency:* The user Is presented with fewer questions during a design session. Questions that previously may have been put to the user to confirm the system's understanding of a particular aspect of the domain may now be satisfied by referring to the appropriate generic model.

o Improved quality of resulting designs: a varying proportion of each design is heavily influenced *by* the appropriate generic model (according to how much of the users domain description can be related to the model). The quality and semantic accuracy of the generic models, and the mechanisms by which the tool interprets the know-ledge represented by these models, are also factors influencing design quality.

Practical demonstration

In this Section, we present a practical demonstration of the use of domain-specific knowledge within a KB-CASE tool. The tool in question, the *Object Design Assistant* (ODA) [17], provides intelligent support during the design of object-oriented databases. It is not the purpose of this paper to discuss either object-oriented databases or the object Design Assistant in depth. However a brief outline of the method of operation of the tool is required in order to illustrate how the domain-specific knowledge may be represented and used during processing, and the associated benefits.

ODA overview and method of processing

The purpose of ODA is to provide support in the design of object-oriented databases. It is intended to be used by a Systems analyst/designer, and assumes some familiarity with Systems modelling concepts and the application domain In question No knowledge of object-oriented databases, object-oriented analysis or design is assumed. The main requirement on the part of a user Is to provide a description of the application domain in the form of a series of declarative statements, and to subsequently provide Information relating to the domain as prompted by the system. The general procedure followed by the major-fly of KB-CASE tools supporting specification acquisition is as follows [1,18]:

- formulate an initial representation of the problem domain.
- use this Initial model in order to generate a conceptual model.
- transform the conceptual model into a design model.

During a design session, ODA follows this three-step procedure. The first step involves the acceptance of a series of declarative statements describing the problem domain, in order to create an initial representation of this domain (the problem domain model). This model then forms the basis of all further processing. The problem domain model is subsequently submitted to a series of refinement algorithms in order to clarify points in question and eliminate inconsistencies.

Once refined, the problem domain model is used to progressively build the object-oriented analysis model. The analysis model represents the

problem domain from a real world perspective in terms of object classes and associated properties. During this step, information is obtained by asking questions of the user and receiving responses. These questions are system generated, based on the system's existing knowledge of the application domain. The responses are used to confirm (or otherwise) the system's understanding of the domain and to further augment the knowledge held.

Once complete, the analysis model is used as a basis for the generation of a suggested design model. The objective of the object modelling step is to take the real world perspective of the analysis model and to transform it into a design which is more relevant to the computer-based perspective. This is a largely automated task, although some interaction with the user of the type previously described may be required.

Representing domain -specific knowledge

To date, there has been very little work carried out in the area of providing KB-CASE tools with real world knowledge and the capability to reason with this knowledge.

Much of the existing work has taken place at the research prototype level and has yet to progress to the point of making an impact on the commercial front.

A comparable work to ODA is the Intelligent Interview System (I2S) [15], 1st in that it supports database design (relational) and attempts to make some use of domain-specific knowledge. The I2S approach makes use of dictionaries relating to specific application domains (for example, library and manufacturing). *Prior* to a design session starting, the user is required to select an appropriate domain dictionary from those available. Each dictionary contains details or verbs commonly found within the application area to which it relates. During a design session, the dictionary is used to construct intelligent responses to user statements and queries. I2S makes use of its domain-specific knowledge to increase Its appear-once of Intelligence and to produce designs of a consistent standard. However, as the dictionaries are used to respond directly to user input the efficiency (in terms of reduced user questioning) is not increased.

Generic models can be used to provide domain-specific knowledge for individual application areas; each model may be thought of as forming a corporate view of a particular application domain. Using this technique,

the user is again required to select an appropriate model prior to commencing a design session. The tool then uses that model as a source of knowledge during the subsequent design process (fig. 31). The use of generic models covering multiple application areas would remove the requirement for the domain selection process. However, this is a highly ambitious proposal and not thought to be a realistic option at this point in time.

In this paper, we advocate the use of a thesaurus-type structure to represent the generic models. Within each generic model, concepts commonly associated with an application are linked together via a series of abstraction mechanisms. This form of representation differs from the 12S approach, In that the expected concepts within a domain are explicated named (for example customer, order) and the explicit verbs of the I2S dictionaries are replaced by abstraction mechanisms. The thesaurus approach allows for each concept to be referred to by any number of associated synonyms where appropriate. For example, a customer may be referred to as a customer, client purchaser, patron etc. The structure used in conjunction with ODA is illustrated by fig. 32. The abstraction mechanisms used to connect the concepts within a domain are those recognised by ODA during processing. i.e. aggregation (A has B), association (A verb-construct B) and generalisation (A is-a B). In addition to these constructs, property attachment statements (i.e. A has property B) are also recognised by ODA. However, properties associated with concepts are not currently represented within the generic models.

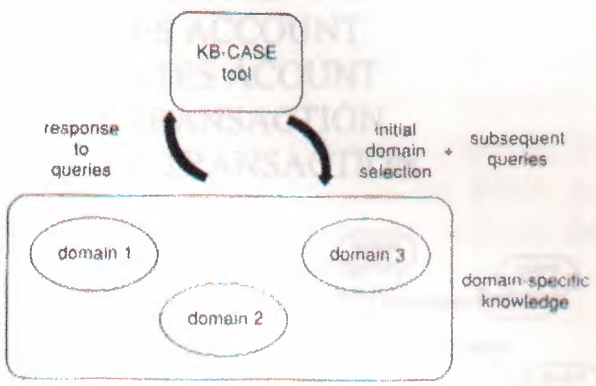


Figure 31. Tool interaction with domain-specific knowledge.

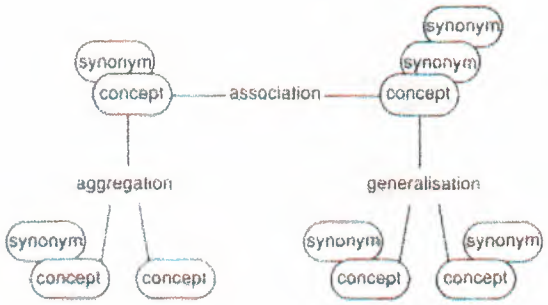


Figure 32. General structure of generic models.

Using the domain specific knowledge

The following series of assertions provide a simple example of the declarative statements expected by ODA The example relates to a banking environment and is restricted In that only sufficient detail required to illustrate the use of the domain-specific knowledge is provided. Information relating to properties etc. has been intentionally omitted for the purposes of clarity (although during an actual design session, ODA would flag this omission and request the missing information).

BRANCH HOLDS ACCOUNT
BRANCH HAS ADDRESS
SAVINGS-ACCOUNT IS A-KIND OF ACCOUNT
CURRIENT.ACCOUNT IS A KIND OF ACCOUNT
CUSTOMER MANAGES ACCOUNT
TAANSACTION UPDATES ACOUNT
DEBIT ISA KIND OF TRANSACTION
CREDIT IS A KIND OF TRANSACTION

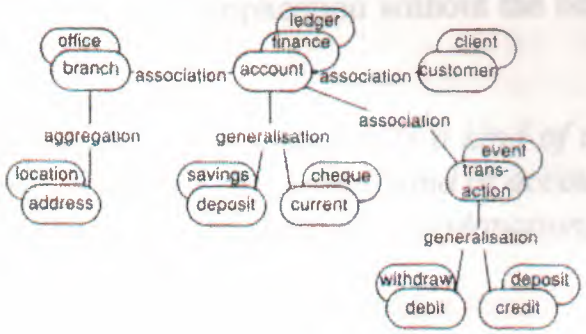


Figure 33. simple generic model for the bank example.

resolved using domain-specific knowledge	not resolved using domain-specific knowledge
confirmation of semantics of relationships between concepts	concepts and/or relationships between concepts not included in the generic model
multiplicity of relationships linking concepts, membership requirements for these relationships (mandatory, optional)	participants in generalisation structures unknown to the generic model
categorisation of aggregation structures (fixed, variable, recursive)	use of terms not present in the thesaurus to describe concepts within the generic model

Table 3. Uses of domain-specific knowledge.

Prior to commencing the design session using the bank example, the generic model appropriate to the domain is selected. A simple generic model presenting the banking domain is illustrated by Fig. 32.

As previously described, the first stage of a design session involves the processing of the declarative satinest in order to create an initial representation of the domain. During this stage, statements are analysed to ensure that they are syntactically connect, but the semantic aspects of the statements are not questioned. The purpose of this stage is to capture the information and to build an Initial representation of the domain as seen by the user (a problem domain mode!). Inconsistencies and potential errors within the problem domain model are then revealed by the refinement algorithms and the user is questioned in order that they may be resolved.

During the second stage of processing, the problem domain model is used as a basis for the construction of the analysis model. Without the use of domain-specific knowledge, this stage of processing is the most time-consuming, as the user is presented with a series of questions based on the problem domain model in order to further augment and to confirm the system's understanding of the semantic aspects of this model. The following dialogue is typical of that taking place as ODA attempts to confirm its understanding of the application without the benefit of domain-specific knowledge.

ODA> *Does the Statement current-account Is a kind of account indicate that current-account is a specialised kind* of *account?*
Please Enter Var N (or H = help, E = explanation):

User> y

ODA> *Does the statement savings-account is a kind of account indicate that savings-account is a specialised kind of account?*
Please Enter Var N (or H = help, if explanation):

User> it

A similar vein of questioning would be pursued with the user regarding the *transaction* concept and its specialised forms, *debit* and *credit.* The use of the generic model enables such questions to be resolved without involving the user The thesaurus approach and use of abstraction mechanisms as a means of linking concepts within the generic model allows the system to exhibit flexibility. For example, the following statements would all be recognised by ODA as being semantically equivalent concerning a single associative relationship between two concepts:

CUSTOMER MANAGES ACCOUNT
CUENT MANAGES ACCOUNT
CUSTOMER IS RESPONSIBLE FOR LEDGER
FINANCE IS MANAGED BY CUENT

Information regarding the multiplicity of relationships linking concepts, the membership requirements for these relationships, and the categorisation of aggregation structures (fixed, variable, recursive) may all be resolved by referring to the generic model. It is recognised that certain questions generated by the system still require user input For example, whether any- other specific types of *account* exist apart from *current* and *deposit.* However, It can be seen that even in the simple banking example, the use of domains-acknowledge results in a reduction in questioning. In larger more realistic situations, a significant reduction in questioning and a resulting saving in time may be achieved.

The use of domain-specific knowledge is not claimed to be a panacea. Indeed, It is recognised that there are certain aspects of the design that are best resolved by user Interaction. Table 2 summarises the areas where the domain-specific knowledge Is of benefit during a design session.

In the case of ODA, the use of domain-specific knowledge during processing achieves two of the associated benefits; the enhancement of the appearance of intelligence, and the improvement in performance. The system appears to 'know' about certain aspects of the application, and the

However, it is recognised that consideration must be given to a number of practical issues. The effectiveness of the approach depends greatly on the accuracy and completeness of the generic models used, and the extent to which idiosyncrasies within a particular domain may be accommodated when compared to the appropriate generic model.

Future required work includes additional testing to obtain statistical feedback on the extent to which performance Is improved during the design of Systems of varying complexity. The extension of the use of domain-specific knowledge throughout the design process is also desirable, in order to influence positively on design quality and the consistency with which such quality designs are produced.

Acknowledgments

The author would like to thank Paul Beynon-Davies, Chris Jones, Mike Lynch and the referees for their helpful comments during the preparation of this work.

EXECUTABLE SPECIFICATION AND CASE

The idea of executing graphical system models Is not new. Several accounts of research in this area are well documented. Despite this, such research is still In Its infancy, particularly, in relation to CASE environments and the practical application of ideas. The paper considers executability within CASE, with a focus on executable specifications. The importance of executability is highlighted, and some of the work in the field is noted. An experimental executable specification tool is presented. The final conclusions drawn take a look into the envisaged future of executability in CASE.

Importance of executability

Harel [1] recognises the importance of executability. He has the futuristic belief that system development tools, which lack powerful execution and code generation capabilities, will all but disappear. He highlights executable specifications as one of the most interesting notions to come out of recent work in systems engineering.

Fuchs expresses views in favour of executable formal specifications [2]. Many of his arguments apply equally to executable graphical

specifications. Tate observes [3]. Practical executable specification of everyday systems poses the problem of somehow combining the kind of informality and flexibility that is natural to most users with the formality of executable Systems.' There is thus a dilemma between the formal specification languages, 'which are not widely applied] and the more popular, but less formal, graphical specification techniques. Work is being carried out to bridge the gap between them [4]. An alternative is to use graphical formalisms which enable executable models to be produced. It is in this interesting area that our work lies.

Several ideas have been investigated for executing graphical models as part of the software process. However, many are not yet integrated into a CASE environment, and so there potential impact on industry has not been fully explored. A goal of CASE technology is to support all aspects of system development Integrated CASE support for the whole life-cycle seems to be on many developers' wish lists [5]. This support should include model executability.

The execution of a graphical specification allows requirements validation to be performed. This finds errors and misunderstandings in the specification, before progressing further into development. Fewer costs are incurred if specification errors are detected early.

Not all specifications are of new Systems yet to be built in fact, such specification tasks are very rare. Usually, some kind of system, possibly computerised, is already in existence. often, one of the initial tasks is to model the existing system in order to propose improvements. Execution of this model, or indeed a component of it, will allow the user to ascertain whether the existing system has been modelled correctly. The executable specification can then be refined to meet the new requirements. The functionality of the system can be observed, through execution, during the refinement stage.

Executable specifications provide a rapid way in which to prototype the functional behaviour of a proposed system [6, 7]. By allowing the external interfaces of the system to be simulated and attached to the executable specification [81, they can exhibit the look and feel of the proposed system.

There is question about whether executable specifications of this kind are truly prototypes, Tate makes some interesting comments on this

[3]. He notes that requirements validation is one of the main purposes of prototyping, and that executable specifications have clear potential in this area. His counter argument is that even If specifications can be executed, they are still fundamentally specifications, With an additional component which aids their understandability. Executable specifications should he viewed as a replacement for their non-executable counter-parts in the software development lifecycle. Prototying [9] involves the development of a program for user experiments. Executable specifications are much more direct than this, as development of a prototype program requires its own design, programming, testing and implementation stages.

In addition to allowing the functional aspects of a system to be validated, execution can also provide a platform on which to study various non-functional aspects of a system. Executable models can be used to analyse timing and performance issues, and for the comparison of the behavioural aspects of separate models [1]. Executability can assist when considering Implementation issues. A system model can be augmented with real world constraints such as a limited number of processors, process priorities and finite execution times. This would enable investigation into the feasibility of implementing the pro posed system, within a given set of constraints.

An executable model of a specification can be incrementally converted through design into a lull implementation by gradually replacing parts of the model with final code. If some of the system under specification already has hardware and software components that are fully implemented, the executable specification system may utilise this code to drive part of the execution. In this way, the model would evolve from a mixture of specification code and implementation code, Into a lull implementation.

This goes a step beyond executable specification and implies that there is an executable model of the system at all stages in its development.

Work in the field

Work has been done on integrating different modelling formalisms. Ward and Meilor [10] and Hatley and Pirbia [1] developed extensions to the data flow diagram (DFD) to allow real-time systems to be specified. Ward also pro-posed a set of rules for executing models created using his formalism [12]. Pulli et *al.* followed up this work by implementing Ward's proposed execution algorithm using Smalltalk inscribed Petri nets [1 3-15].

Reilly and Brackett [16] present a useful account of the requirements of an idea tool to support the execution of structured analysis models. They also built on Ward's ideas and investigated a method of specification execution, involving the translation of a structured analysis model directly into an OPS5 program. The 0PS5 program is executed in an interpretive manner, and the executions are examined using the trace facilities provided by the 0PS5 run-time support system.

Lea and Chung [6] have done work on rapid prototyping, taking an executable specification approach. They consider each DFD to be a set of regions of bubbles and data flows, which correspond to system service functioning. Their technique, in contrast to ours, does not involve the coding of primitive process functions in traditional pro gramming languages.

Harel's STATEMATE system [17] is a set of tools which support the analysis, design and documentation of complex reactive systems. The temporal reactive behaviour of the modelled system Is depicted using the graph!-cal language *Statecharts* [18]. STATEMATE allows specifications to be executed in a step-by-step interactive manner, or in a non-interactive manner, 'via programmed executions. The executions allow reachability, non-determinism, deadlock and transition usage tests to be performed.

At Hull University, 'we have implemented a demonstrator system that allows data-driven functional models to he executed and animated. A particularly novel aspect of this work is the provision of execution by adding functional language code to the graphical specification model. Experience gained from this system has led us to develop a new visual formalism, with real-time extensions, for use in the next version of the system. An algorithm has been developed for executing specifications which use the new formalism.

The possibility of extracting an object-oriented model from a data flow- specification has been shown previously [19]. Ward also shows the relationship between object orientation and traditional structured analysis and design [20]. Our experience with modelling systems has shown that when modelling requirements in the traditional way' using DFD's, the partitioning of functionality is often object-based. Current research at Hull University is investigating the possibility of maintaining a consistent object model across the whole system development lifec-de. The ability to execute such a model would fadlitate the use of incre mental substitution and testing of each new or modified object, as the system evolves.

EGS project

Project background

The aim of the Executable Graphical Specifications (EGS) Project was to develop an executable graphical specification system which could he used as a vehicle for experimentation Into the provision of executability within CASE tools. The underlying modelling formalisms and their links to the method of model execution were an area of Interest An additional aim was to investigate the different way's of providing model execution and to consider the use of excutable models throughout the lifecycle.

A tool was required to enable experimentation with the various aspects of executable specification systems. A brief study of meta-CASE technology was undertaken, and specifically the IPSYS [21] meta-CASE system was investigated, as a possible aid to building the tool. However, current meta-CASE technology did not appear able to provide, in a convenient way, the flexibility required by our tool, and so we decided to build It more or less from the beginning.

The task of building such a tool was quite complex and involved the combination of various technologies. A two tier approach was taken to the development The plan was to build an initial system, with a simple in-built formalism and execution scheme, In order to test our general ideas. Case studies performed on this system and experience gained from its development would then be used to build a second, more substantial tool.

Building the initial system also identified the desirable generic aspects of building CASE tools of this nature. Such issues as notation-independent editors, methods of providing execution, and keeping the execution language separate from the execution method emerged as Important considerations. Clearly, these form the basis for a meta-CASE tool to aid the production of tools that support diagram execution.

In this paper, we describe the initial executable graphical specification tool which was developed and implemented. As this system represents phase I of the Executable Graphical Specifications Project, we refer to it as EGS 1 -Work on this initial phase of the Project is now-complete. Efforts have since moved to an upgraded version of the tool.

EGS 1 tool

EGS 1 allows functional models to be built using an inter-active editor. The graphical formalism supported by the system resembles the DeMarco DFD notation [22]. The models built consist of a hierarchical set of data flow diagrams. These contain processes and data stores with data flow- interconnections. The highest level diagram in the hierarchy (the context diagram) also contains external entities. Processes are either primitive or non-primitive. The non-primitive processes expand out Into lower level DFD's.

The DeMarco [22] notation is well understood and has been accepted by many software engineers. It forms the foundation from which several extended notations were developed, and it fits into a simple execution model. It Is for these reasons that the notation was chosen for EGS 1.

The user augments the models with code specification dements. This enables the models to be executed in an Interpretive manner, thus allowing the functionality of the specification to he observed.

Fig. 33 shows the context diagram of the specification of a video cassette recorder (VCR) control system. The role of the VCR control system is to accept user commands and generate various signals to control the head mechanism, display and motors of the VCR. The main components of the system are a tuner manager, function control manager and timer manager. The tutor manager enables the channel to be set during normal operation and recording. The function control manager determines the required motor and head position settings for a given operation. The timer manager facilitates the preset (programmed) recording operations. The VCR was one of several small case studies used to evaluate EGIS 1. Aspects of this study are used here for illustration.

Model executability is provided by the Gofer functional language, which is a subset of Haskell [23]. Gofer code is used to specily the behaviour of primitive processes, and the type of data associated with each flow and store. Functional languages are particularly appropriate for the specification of primitive processes [24]. The EGS 1 formalism views primitive processes as strictly functional: thus there k no requirement for them to have state. Any stored state information must be abstracted out and expflctly rep resented by data stores. Functional languages provide a high level of expression, thus redudng the verbosity of pmcess

specifications. The declarative style of functional languages ts also appropriate for expressing the relation-ship between inputs and outputs. Consider the level 0 DFD of the VCR specification shown in Fig. 34. Some processes. In this diagram are non-primitive and expand out into lover level DFDs. GET STATUS, however, performs a simple operation and thus is primitive.

The GET STATUS process is triggered by the receipt of a data packet along the switchPos flows. We can see from Fig. I that this packet has been generated by the STATUS SWITCH external. The switchPos flow- has VcrStatusType in its underlying textual definition VcrStatusType is not a standard Gofer type. It is a user-defined type 'with the following definition:

data VerStatusType =

STANDBY I ON TIMEHIDLE TIM ERACTIVE

VerStatusType has been declared in this way because it is used by other elements of the specification model. EGS 1 allows global types and functions to be declared for use by other functions and types. Such global elements are stored in a general global declarations area.

The data store VcrStatus also has VerStatusType in its underlying textual definition, Indicating that the value it stores is of the same type as the data packet carried by the switchPos flow. The update flow leading from GET STATUS has mt in its textual definition, 'which is a standard Gofer type.

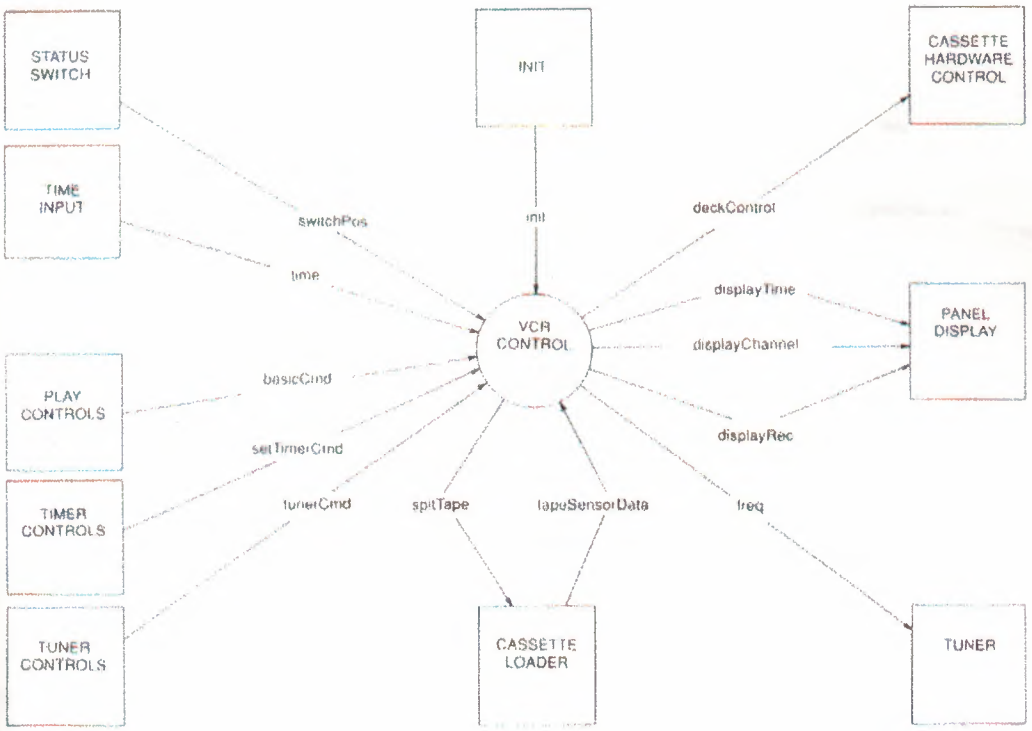


Figure 34. VCR context diagram.

The task of GET STATUS, when triggered by a change in status switch (an incoming packet on switchPos), is to update the current VCR status and indicate to the TUNER MANAGER when the device has been switched on. The underlying textual definition of GET STATUS is

= <verStatus = switchPos, update = value>

where {
value switchPoa = = ON = 1
I otherwise = 0
}

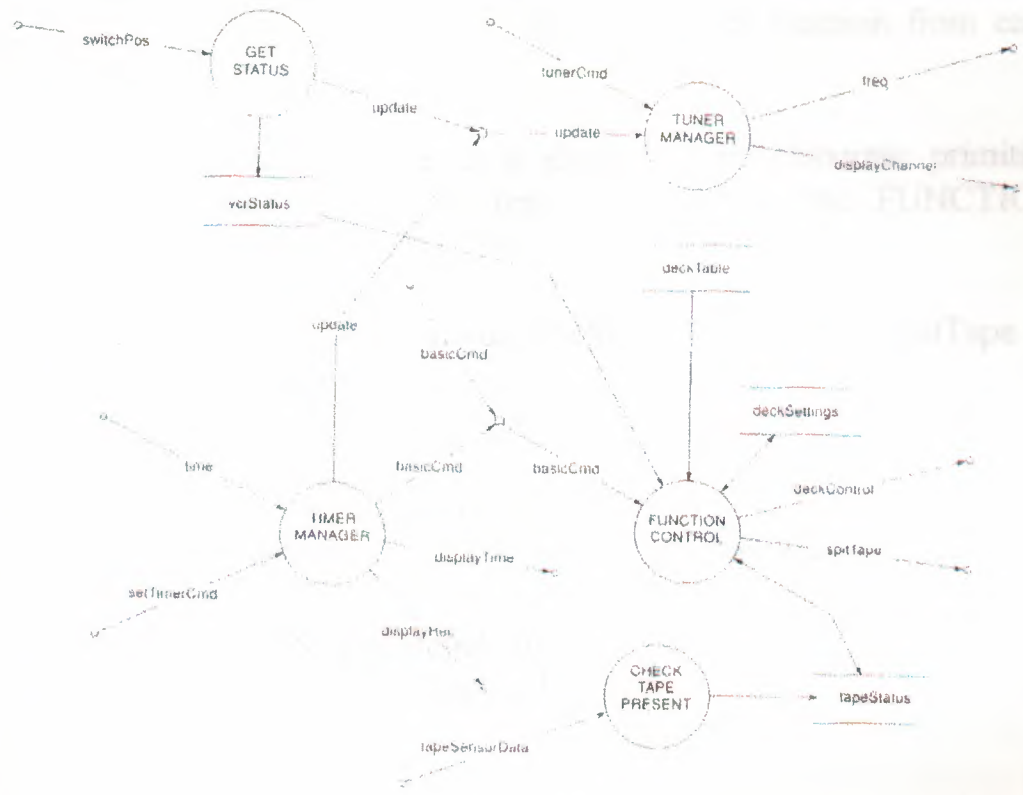


Figure 35. VCR level 0 DFD.

Note that this code is not written as a pure Gofer function. No parameter list is present Such information is of no concern to the user, and so is hidden. The textual definition simply specifies the value of each output in terms of the input values. Here the update flow and the VcrStatus store are defined in terms of the switchPos input flow.

Each output of this particular process is dependent on the input value. This does not have to be the case for all processes. Outputs from a process could be simply assigned constant values. Processes whose sole task is to initialise data stores often fall into this category. The process specification shown here is also quite simple.

However, complex expressions, which possibly call globally defined Gofer functions, may be used to calculate the output values.

From the formalism point of view, each primitive process is regarded as a pure function, i.e. a mapping of input data flows to output data flows. Model execution is based on converting each primitive process specification into a self-contained Gofer function, which can be executed

to simulate the processes' behaviour. One of the tasks of the model compiler subsystem of EGS 1 is to derive a Gofer function from each primitive process.

The following example is a slightly more elaborate primitive process specification. It is the text contained in the FUNCTION CONTROL process shown In Fig. 36.

```
=     <deekControl = deckVal deckSettings = deckval, spitTape =
spitVal, tapeStatus = tape>

where
spitVal ((vcrStatus = = ON) && (tapeStatus
- TAPEPRESENT) && (basicCmd = = EJECT))  1 otherwise = 0;

tape ((vcrStatus = = ON) && (tapeStatus
  - = TAPEPRESENT) && (basicCmd
  - = EJECT)) = EMPTY
  otherwise = tapeStatus;
  fn x y - head ((b,c,d,e)j(a,b,c,d,e) <- y1a
   deckVal ((verStatus = = ON)
        &&((basicCrnd = = PLAY) II (basicCmd
     = = STOP) II (basicCmd = = REC) I (basicCmd = = REW) p
(basicCmd
                                = = FF))
        &&(tapeStatus = = TAPEPRESENT))
                  = fn basicCmd deckTable

         ((verStatus = = TIMERACTIVE)
         &&(basicCmd = = TREC)
        &&(tapeStatus = = TAPEPRESENT)) = fn basicCind deckTable

             ((verStattis = = TIMERACTIVE)&&(basicCmd = = TSTOP)
             &&(tapeStatus = = TAYEPRESENT))
          = fn basicCmd deckTable

         otherwise                =                (ZERO,RETRACTED,
FORWARD,TUNERTOSCREEN)
```

All flows are discrete in the formalism. Therefore, the view of execution is that of discrete packets of Information flowing down data flows. Primitive processes can execute when at least one packet is present on each of its input data flows, and all stores used as input to the process have defined values. An uninitialised data store stops a process from being considered for execution.

The execution of a primitive process causes one packet to be consumed from each of its input data flows. The values stored In its Input data stores are used for execution, but are not consumed or updated. The execution of a process may produce zero or one packet on each of its output data flows, and may also update the value of any data store which is connected to it via an output flow.

Packets wait on data flows until they are consumed by an executing process. There is no limit to the number of packets 'which can watt on a data flow at any given time. Packets can thus queue along data flows in the order in 'which they arrived. 'When a process with a queue of packets on one of its input flows executes, it consumes only the first packet in the queue. The rest remain on the queue and are consumed by future executions.

A process may become executable either by receiving a packet from an external entity or by receiving a packet produced by the execution of another process. It should be apparent that the injection of packets from the external environment can set off a chain of subsequent process executions.

Execution of EGS 1 models is thus driven by data and occurs in a stepping manner. The execution of each primitive process is assumed to be a single step. No sophisticated scheduling algorithm is employed. If several processes are eligible for execution simultaneously, the one to be executed next is chosen at random.

Once a specification model has been compiled, it can be executed and animated by switching to run mode. In run mode, steps are executed in response to user requests. In addition to single steps, the user can instruct the system to execute for a given number of steps or execute until no more processes are runnable. Break points can be set on processes, so that when they are scheduled to execute, the system halts. This provides a convenient mechanism which enables the user to cause the system to rapidly reach a state of interest before looking at the model execution in finer detail.

The user provides input packets from the external environment by inter-actively adding packets to the external input flows of the model. The mouse is used to select the external input flow onto which a packet is to be added. On selecting a flow, the user is prompted to type in the value of the new packet. This can be done at any time during model execution.

Packets intended to be output by the executing system, to the external environment queue along the external output flows.

The model is animated to show the flow of data through the system and the primitive process executions which relate to this data flow. The user can navigate between diagrams in order to observe the desired components of the executing model fig. 35 shows the level 0 DFD of the VCR case study in run mode.

Note the difference in appearance or the diagram to that shown in Fig. 36. In run mode, primitive processes that are not currently executable have slightly thicker borders than non-primitive processes, which are not executable elements in their own right FUNCTION CONTROL and CHECK TAPE PRESENT are two primitive processes that fall into this category. GET STATUS Is also a primitive process, but it has an even thicker border to signify that it is currently executable.

A similar convention is used to represent flows. In run mode, the name of each flow has a number appended to it which represents the number of packets currently queueing on the flow. In Fig. 37, this number is 0 for all flows apart from switchPos. Any flow with one or more packets queued on it is represented with a thicker line than flows with no queued packets.

It should be noted that the animation described here was designed for monochrome displays. If a specification Is executed on a colour display, different colours are used for animation instead of varying line thicknesses.

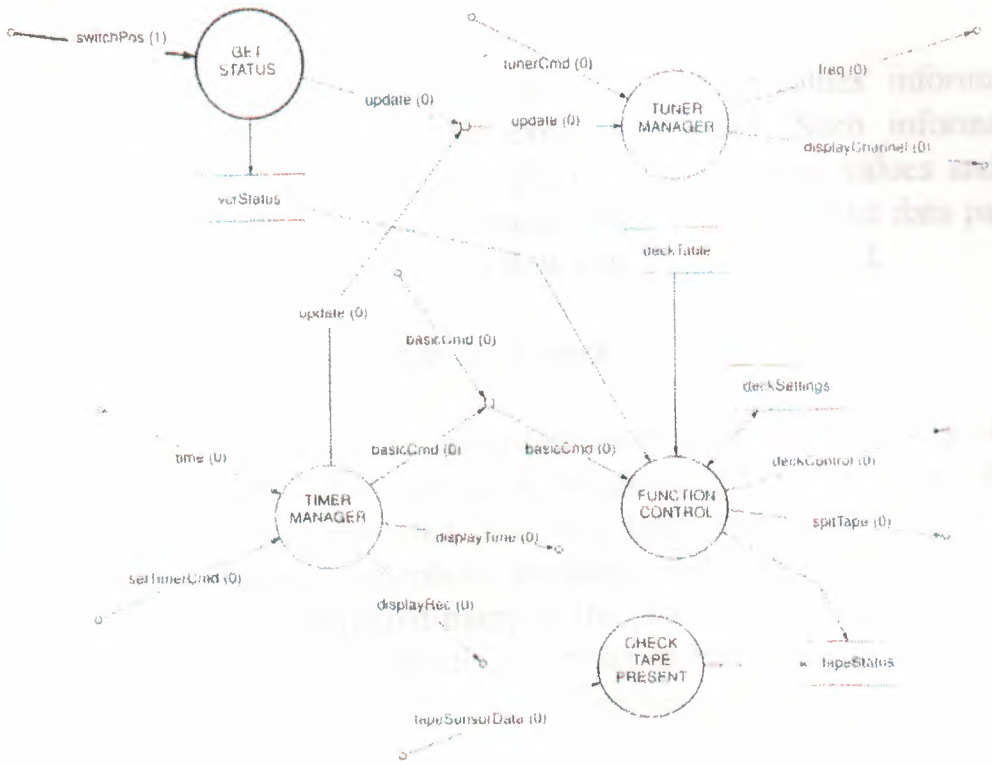


Figure 36. VCR level 0 DFD in run mode.

The role of animation is to visualise execution of the specification model. It aims to make available to the user the execution Information of Interest. The animation provided can be considered in three separate categories. The terms 'Static visualisation', 'Dynamic visualisation' and 'Historical Interrogation' are used to refer to these categories.

'Static Visualisation' refers to the state information which can be obtained when the system is in a static state. It includes such aspects as the current state of a process (whether it is executable or waiting for input) and the state of a data flow (whether it currently has packets queued on it, and if so, the number of packets queued). Such information Is displayed visually on the diagrams. The user can also interrogate any data store in the executing model, in order to observe its current contents.

'Dynamic visualisation' refers to the dynamic behaviour of the specification model as it executes. It refers to the flow of data packets around the system and the various transitions which occur in the process execution states. This is a combined picture of the various static visualisations, within the context of a dynamic execution. It cannot be illustrated

in this paper, as it is in essence a series of frames which form a 'movie' of the executing model.

'Historical Interrogation' refers to the 'previous values' information which can be obtained from the executing model, Such information includes the last set of input values, the last set of output values and the last execution time of a primitive process. The value of the last data packet which passed down a particular data flow can also be obtained.

Conclusions from EGS 1 and further work

The relatively straightforward formalism and associated execution scheme adopted by EGS] allows quite complicated models of data-driven systems to be built and animated. We have built executable models for a package routing system, a telephone exchange and a VCR Clearly, EGS 1 is limited, but it has highlighted many of the problems and questions to be asked about executable specification systems. It has allowed us to look at the development of such tools from an experienced practical viewpoint.

Data store initialisation was highlighted as an important issue. Sometimes data stores need to be initialised with values before the rest of the specification can commence execution. Although omitted from the diagrams seen so far, EGS 1 adopted an explicit approach to data store initialisation. EGS 1 specifications often include processes dedicated to the initialisation of data stores. Fig. 39 shows the level 0 DFD 'with data store initialisation processes included.

It could be argued that the initialisation of data stores should be done explicitly in this way, as stores are internal to the specification and their initialisation needs to be considered. indeed, the developer of a specification should be able to feature store initialisation explicitly if desired. However, such initialisation often clutters the model and detracts from the rest of the specification. It is for this reason that an alternative provision is required for the initialisation of data Mores. Initialisation code could be associated with each store, within a component of its textual definition, just as the type of data contained in a store is currently specified. In this way, the initialisation of stores is not ignored, but it does not have an over-emphasised visual impact on the specification.

EGS 1 only caters for the data flow components of system modelling. The examples illustrated show the DeMarco [22] type formalism which is currently supported. However, as the notational aspects

are rule-based, EGS 1 is independent of the actual visual formalisms used. The system could be tailored to support a different notation, as long as it fits the general data flow paradigm.

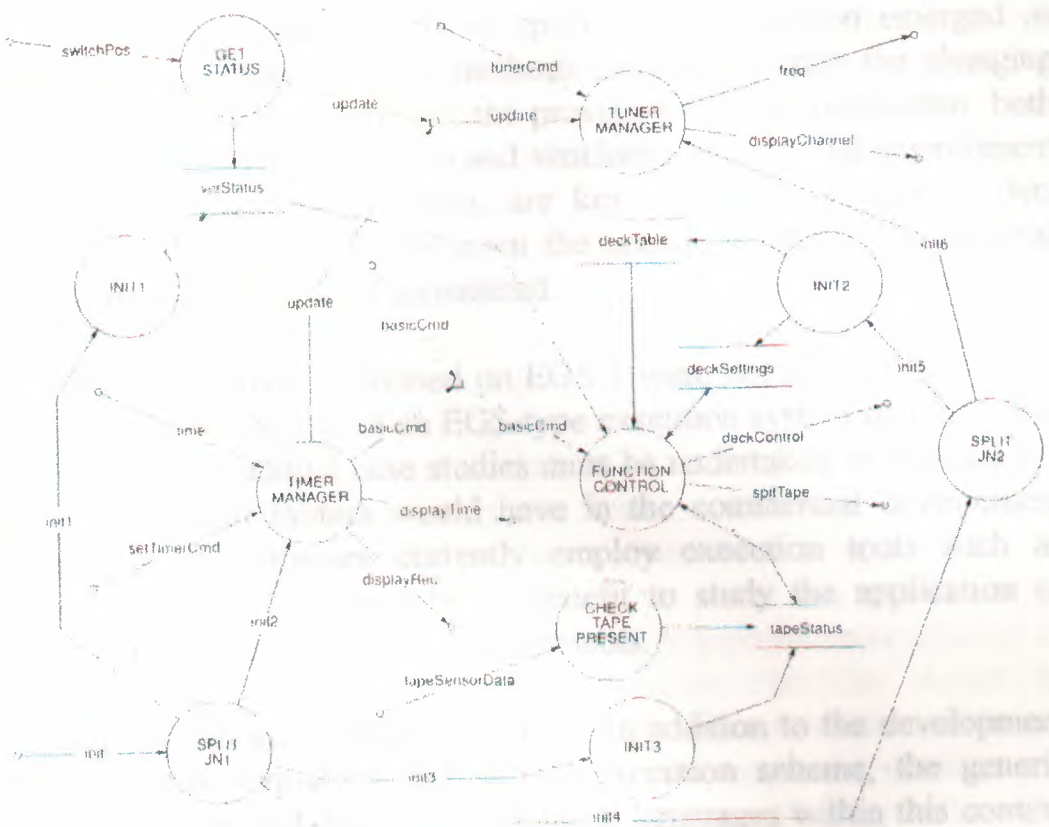


Figure 37. VCR level 0 DFD with explicit store initialisation.

One of our goals is to provide a more general execution tool which allows specifications with real-time aspects to be modelled and executed, A tool of this nature requires a more elaborate formalism, suitable for specification of real-time systems. Such a formalism has been developed in close relationship with a more elaborate execution scheme. This Work is documented elsewhere [25]. Time Is a key aspect of real-time systems modelling. By considering time in the execution scheme, issues of process scheduling and the temporal relationships between execution steps become important. These Issues are accounted for in the new execution scheme.

The EGS 1 editor allows diagrams to be entered and stored in the repository (static model) component of the system. The model compiler generates the executable version of the model from the static model data. it is thus clear that if the data captured by an existing CASE tool could be

translated into the EGSI static model format, other CASE tools could be used to capture EGSI models. This would be an additional useful feature and has been noted for future consideration.

The user interface aspects of specification execution emerged as important for consideration. The methods used to illustrate the changing state of the executing system, and the provision for user interaction, both In terms of directing the execution and simulating the external environment of the system under specification, are key issues. Simulation of data transfer across the boundary between the specification and the external environment also needs to be considered.

The case studies performed on EGS 1 were not substantial enough to Investigate hilly the use of an EGS-type execution system on real world projects. More substantial case studies must be undertaken to examine the impact this kind of system would have in the commercial environment Although some companies currently employ execution tools such as STATEMATE [17], it would be of benefit to study the application of EGS-type systems to Industrial-based projects.

Our current work builds on EGS 1. In addition to the development of a new visual formalism and related execution scheme, the generic aspects of the tool and the use of functional languages within this context have been further researched. These recent developments are Incorporated into EGS2, which is an upgraded executable specification tool, soon to be released. Planned future work includes several extensive case studies which employ EGS2 in realistic software engineering situations. Such studies will investigate the practical use of the system and highlight further scope for improvement.

The Dynamic Modelling of Embedded Systems Project, undertaken in collaboration with UMIST and Staffordshire Universities and sponsored by SERC, is expected to provide an industry-based test bed for our work The project is concerned with looking at a range of methods for modelling dynamic systems. The modelling methods to be investigated include EGS2 [25], STATEMATE [17] and the Co-Design method from UMIST, The Project will be looking at several large industrial case studies extracted from GEC, ICL and BAs.

Future of executability In CASE

CASE tool support is required for the execution of the various models, produced in the software development lif6 cycle. The execution must be frilly integrated into all levels of the life-cycle. Further research is required Into methods of adding execution to graphical models. This should be considered in conjunction with investigation into the integration of the differing modelling formalisms.

Current CASE generally provides good support for the diagram creation and information storage aspects of system modelling, with the use of diagram editors and repositories. Similar support is required to facilitate model execution. CASE tools which provide support for model execution place heavier demands on the repository servers. This is particularly relevant to the compilation and animation of models. As with standard program compilers and run-time support Systems, time is of the essence; thus, we must stative for more responsive repositories.

The large amount of effort required to provide executability in CASE tools is hindering the research process. Facilities should be provided which allow diagrams consisting of various different notations and pieces of text to be easily integrated and executed. CASE tools supporting such model execution must be as general as possible. The underlying tool should be independent of the graphical for malisms employed. The user should not be limited to any particular language for providing the execution. The compilation process involves the use of several different compilers, if models incorporate different languages. The model compilation and execution process should be independent of the languages chosen to facilitate the execution.

If truly generic execution Systems were available, their scope for use would be vast it is expected that issues raised by research into meta-CASE technology may con-tribute to the development of the generic aspects of such systems. Diagram compiler generators, which are capable of generating diagram compilers from rules which relate to particular graphical formalisms, are expected to feature in the future of executable specification technology.

Various levels of animation will be required. The end users view of the executing model must be considered as well as animation of the behaviour of low-level model components. If executable models consisted

of specification, prototype and final implementation components, the Issues to be considered In animation 'would be extended yet further.

If the final system is delivered as a package of analysis models, design models and implementation software, set in an executable CASE environment, the customer would have a seamless, executable model of all stages of a system's development. This would have obvious advantages if the requirements of the system were to change at a later stage.

So far, research Into model execution has been focused on dynamic/reactive systems. We feel a more general approach must be taken that allows the execution of various different kinds of model, possibly usIng a combination of different visual formalisms. Equal consideration could then be given to highly data- or functional-oriented systems. To facilitate this the tool must be adaptable to different formalisms.

Executabilily will form a predominant role in the future of CASE. Executable graphical specifications are an important concept, but form only a fraction of the potential for execution within CASE environments. if CASE tools are to provide support for maintaining an executable model all the way from the initial requirements stage through to implementation, it is unlikely that the division between the various stages of the traditional lifecycle [9] will be maintained. Further research into alternative life-cycles, which include model execution is thus required.

CASE SUPPORT FOR COLLABORATIVE MODELLING : RE-ENGINEERING CONCEPTUAL MODELLING TECHNIQUES TO EXPLOIT THE POTENTIAL OF CASE TOOLS

To date, CASE tools have generally been built around pre-CASE analysis and design techniques. The paper argues that more benefit would he obtained If analysis and design techniques were 're-engineered' so as to make the best use of the capabilities offered CASE tools. Techniques such as entity-relationship modelling and data flow diagrams have successfully been transferred to the CASE environment, with significant administrative and clerical benefits. However, these techniques were not necessarily designed with automation In mind, and their CASE Implementations have failed to adequately address important aspects of the modelling process, such as communication, collaboration, and the application of past experience. Ways In which these aspects can be supported pro-actively by CASE tools are given, using examples from a prototype CASE tool.

How could CASE* tools *support the analysis process?

For the purposes of this discussion, the process of Systems analysis Is considered from several perspective's:

- as a process or communication, capturing relevant information about requirements.

- as a process of collaboration or negotiation in which requirements are jointly 'developed'

- as the application of past experience in creating, reformulating and emulating models.

In each case, existing or potential support from CASE tools is discussed. No one perspective offers a complete picture of the role of Systems analysis, but each has some thing to offer in helping to put the use of CASE Into its proper perspective.

Analysis as communication

Traditionally, the earliest stages of information Systems analysis have been referred to as 'requirements capture'; the idea being that requirements for Information Systems exist Irrespective of what is possible or feasible 161. The analysis process consists of documenting requirements and then investigating feasibility or looking at costs to determine what will be implemented.

The process of documenting requirements is seen as relatively straightforward, because the requirements are either known or implied by an existing System. Consequently, top-down methods are often used, as they also imply a relatively straightforward documentation of requirements, by decomposition of high-level requirements into lower-level ones. CASE tools readily support this approach; indeed, many enforce it For example, one currently popular CASE tool [7], widely used for SSADM projects, has only recently added limited support for the relocation of processes from one level of a DFD to another. This type of change is almost inevitably necessary unless requirements are fully understood before the diagrams are constructed. In other words, it has implicitly been assumed that the analyst gets each diagram right first time, and that there is little need for 'rehashing' a diagram once it has been built. Constraints of this

sort mean that many CASE tools offer poor support for an iterative way of working.

Owing to the overheads associated with amending models, CASE tools have tended to take on a narrow role; the documentation of models that have already been developed on paper. The difficulty of modifying models held in many CASE repositories means that the tools will not be used to capture models as they *are being developed.* how could CASE address this problem? One solution is to provide quick and easy ways of changing models. In the CASE tool mentioned above [7], to move a process between levels is (arguably) almost as hard as deleting the process and adding it elsewhere; data flows etc. must be individually reinstated. Moving a process between levels is, however, conceptually a very simple operation; employing simple user interface design principles, the CASE tool could offer a correspondingly simple 'drag-and-drop' style of manipulation, allowing the process to be moved with the minimum of effort [8].

Mother way in which CASE tools can support an iterative approach is by minimising the level of formality required in a model. The assumption of top-down decomposition fails to take Into account the fact that only fragmentary information may be known at the early stages of analysis. If the CASE tool protests because certain data flows go nowhere] or certain data attributes are not assigned to any data entity, or a process is unattached to any other objects, then It is making unrealistic demands of the analyst. Once again, it is forcing the analyst to use the tool to document models, rather than to develop them.

Analysts as collaboration

A more up-to-date view of today's systems development is as a process in which developers and users work together to explore possibilities and to find out what is mutually acceptable. The concept of partidpative design epftomises this view. The IT professional's role has therefore moved in some 'ways towards that of a 'facilitator'. It is still important to gather information, but this view of systems development emphasises the need for bidirectional communication between analyst and user. Being able to support thls collaborative activity is not necessarily easy.

First, the analyst and user are likely to have very different skills. IT-oriented tools can easily confuse and disorientate a non-technical user. What is needed are tools that manage to bridge the gap by hiding 'technical' concerns, while allowing full expressivity to capture and represent important information. Secondly, collaboration is an inherently unpredictable activity. Ideas emerge and are bounced around; they are combined, changed, recombined or discarded. Finding ways of providing support for an undefined task is tricly. Nevertheless, it is important that we do offer support for the collaborative process. In the following, we look at some ways In which appropriate support might be provided, with examples from a prototype CASE tool, currently under development, called visual Modeller.

Hiding The 'technical' aspects of modelling:

Many CASE tools present a rather formidable 'technical' face to their users. For example, the representation of common diagrammatic modelling techniques is extremely logical, to the mathematically minded IT person, but could hardly be described as appealing, evocative or Inspiring to non-IT users. Fig. 35 gives one example, an object diagram, constructed using boxes, connecting lines, arrows and text Entity-relationship diagrams use boxes, lines and text; data flow diagrams use circles, arrows and text These techniques were created so as to be easy to draw using pen and paper. Yet today's computer hardware can display graphics with ease. Perhaps models could be created from *truly* graphical primitives; ones that resemble the things they represent.

The visual Modeller prototype addresses this issue In three main ways.

- *Use of icons for ease* of recognition: icons, and not just text, are used *to* represent model components, facilitating an approximate but immediate visual interpretation. In entity-relationship modelling, entities as different as customers, orders, departments, and documents are rem resented using the *same* symbol. In visual Modeller, customers are represented using quite distinct icons from documents, for example. Fig. 34 illustrates a model being constructed In Visual Modeller. The choice of Icon is important, and therefore Visual Modeller allows appropriate Icons to be selected from a library.

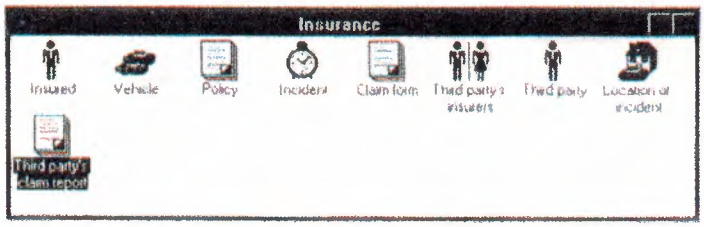


Figure 39. Simple model.

o *Specifying relationships through direct manipulation:* in entity-relationship modelling, and many object-oriented analysis techniques, aggregation relationships are shown as explicitly drawn connections between objects, with associated labels, cardinalities etc. If the relationships need to be amended, connections must be deleted and others added. The multitude of relationships on a typical entity-relationship diagram can make interpretation a real challenge. In Visual Modeller, manipulating a model and Its components is almost as simple and intuitive as manipulating windows and icons; aggregation relationships are created simply by placing icons within windows.

o *Implied relationships:* In entity-relationship modelling, most relationships are drawn explicitly. In Visual Modeller, however, relationships are not shown; instead they are implied. Subtype/supertype (or inheritance) relation-ships are Implied by the physical appearance of icons; for example, two model components that both have 'person Icons both represent people or specific types of people (such as employees). other associations are Implied by the naming of components.

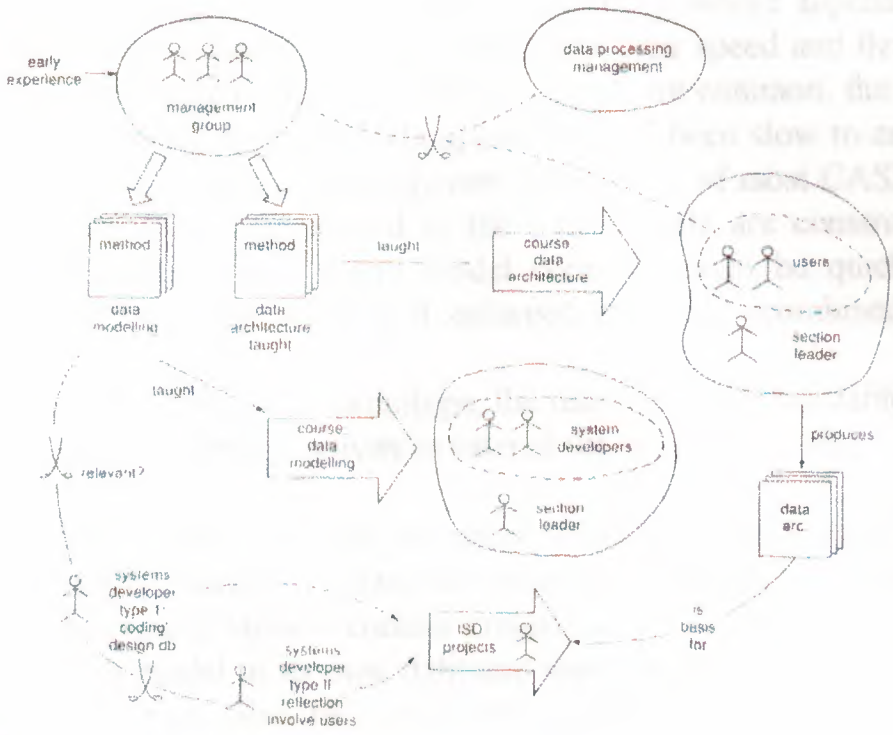


Figure 40. SSM rich picture [10].

It should be stressed that the idea of using richer, conic representations for models is not new. For example, the rich pictures used In the soft systems methodology (SSM) and its derivatives [9, 10] are a well established way of rept It is intended to create a suitable unisex version of the 'person kon when time allows resenting a business situation in a graphical form intended to be especially accessible to non-technical people. For comparison, a rich picture is reproduced in fig. 40. Work is under-way on providing CASE support for the creation and manipulation of rich pictures. In tact' many CASE tools already offer limited 'system diagram' features, in which simple icons, usually of people, documents, equipment etc., can be put together. The reason often given for mailing this sort of facility available is that diagrams with pictures are more easily understood. We wonder why. if this is the case, the idea has not been applied to the 'serious' modelling techniques.

Providing true modelling flexiday: despite its inherent unpredictability, the one certain *thing about* modelling is that models change. Often, *they* need *to* change quickly as ideas develop. If the CASE tool cannot 'keep up with the pace of model development, then it must be relegated to a passive documentation role. The 'brainstorming' style of modelling is fully

supported in joint applications design (JAD) [11], where flipcharts and white boards are the preferred tools, chosen for their speed and flexibility. Although JAD and JAD style modelling sessions are common, the Idea of using CASE directly during modelling sessions has been slow to catch on: "we argue this is because of the inherent inflexibility of most CASE tools. Flexible CASE tools truly suited to the way models are constructed in reality would allow' models and model components to be quickly and easily built, revised, combined, split, enlarged, reduced, recombined etc.

In the visual Modeller prototype, the inherently unpredictable nature of collaborative modelling activity is catered for in several ways.

- *Reduced construct set.* first, the set of constructs used *to* build models is very simple; there is no clear-cut distinction between a 'model' and a model component Models contain components, but each component is effectively a model in its own right and may contain other components. Components may also be connected to other components; but this connection is implied, by position, appearance or name, and so there are no messy connections between components to worry about when components are reused, moved, deleted or added.

- *Simple model manipulation:* to enable model-building from components, a form of object-oriented model has been employed in which aggregation is used to build components from other components, much as a class in an object-oriented system can be constructed as an aggregation of other classes. Fig. 39 illustrates a simple model component containing several other components. Components are added or moved simply by dragging and dropping, as In a classic GUI interface; the sub components of a component are then automatically added or moved.

- *Support for 'unstructured' analysis:* the early stages of modelling often yield confused, fragmented and contradictory Ideas. It is useful to capture these ideas and to follow them through; this is the essence of 'brainstorming'. A tool that forces consistency and logicality onto what is essentially an unstructured process is therefore unsuitable; yet few CASE tools permit the capture of models In their early, unstructured form. Many of the ideas developed at early stages, through refinement, form the basis of later model versions. It is wasteful to discard the results of earlier stages simply because they are not rigorous enough.

Visual Modeller caters for this by allowing the analyst considerable latitude, within the framework of windows and Icons, to construct arbitrary models. The models need not mean' anything in particular; preliminary models often contain objects purely as placeholders or reminders of issues to be resolved. The important process of model refinement, which proceeds as the analyst begins to fully understand the business area, is therefore supported. As changing models is simple, there is no penalty attached to starting with a 'wrong' model and then making it 'right'. Fig. 37 illustrates a model at an early stage of this restructuring process.

Analysis as the application of past experience

In this third and final perspective, the idea of gathering or negotiating *requirements* is seen as less important than the need to *understand* a business area before any computer-based information system can be contemplated. Gaining such understanding is a necessary precursor to any discussion of requirements, and the creation of models is a formalised way of understanding a situation and of demonstrating understanding. A model can be checked to ensure that it truly represents the area of concern; it can be used as the focus of debate, and its structure can be used to guide the information gathering process.

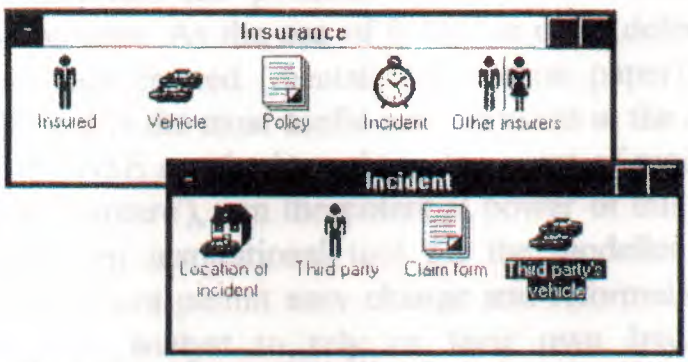


Figure 41. Restructuring a model.

How can CASE tools assist In the process of under-standing? To help in answering this question, we must decide what we mean by 'understanding'; a simple model of human cognition helps us to understand this process (Fig. 42). Insights from the construction of artificial neural networks that mimic the operation of certain parts of the brain appear to supine this model, According to the model, new experiences are interpreted in the light of past experience; relevant memories are conjured up, by association, in the process known as 'cued recall', When a new

piece of knowledge is committed to long-tend memory, ft is always in a particular context; memory Is essentially *associative,* Facts are not learned individually and in isolation; they form a complex web of Inter-relationships and connections. Thus, the process of understanding a situ-ation involves the interpretation of the situation In the light of what has gone before.

What are the implications of this simple model for the designers of CASE tools? First, we see Immediately that CASE can provide useful support for human memory, as a CASE tool can manage enormous amounts of information; the repository can be seen as a powerful 'memory for storing and associating facts, provided that they are first placed into a sufficiently structured form. This Is indeed the approach taken in most CASE tools, where the role of the tool is primarily to ad as a database for meta-data. However, if we look again at our model of 'understanding', we also see that learning and understanding new ideas are intimately linked to the *retrieval of arising knowledge.* Generally, we can correctly interpret a new situation provided that we can recall relevant past experience about related or even similar situations. Making a creative leap without experience to draw on (in other words, lateral thinking) is a rare skill.

In the context of modelling a business situation, it can be argued that the use of CASE has the potential to interfere drastically with the 'understanding' process. As the use of CASE is often deferred until *after* the model has been created (mentally, if not on paper), the powerful 'memory is not used in the most useful way; to assist in the *construction* of models. Only if CASE can be brought to the point of model formulation (or 'requirements capture'), can the potential power of this memory as an associative and even inspirational tool for the modeller be unleashed. CASE tools that do not permit easy change and reformulation of models therefore force the analyst to rely on their own Intuition and past experience; effectively, the analyst must keep the contents of the CASE repository in their own mind while modelling. otherwise, we are faced with the onerous task of modelling every situation afresh and then trying to integrate a possibly incompatible model back Into the repository.

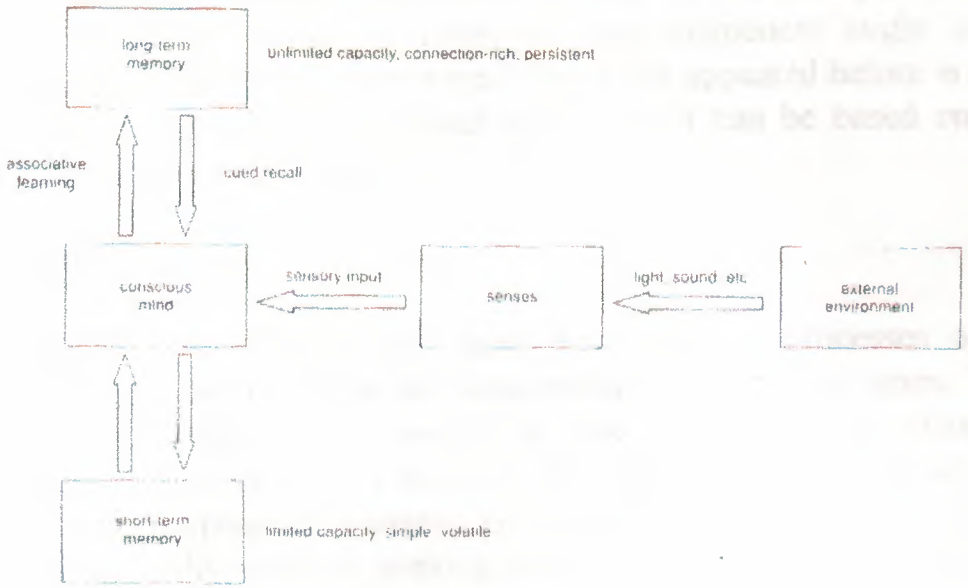


Figure 42. Simple model of human cognition.

Thus, we conclude that allowing easy change and reformulation is not enough. *To be truly useful In the process of creating models, a CASE* tool must offer positive guidance and help; or we are not to model afresh every time, then it is useful to be reminded of what exists already. For example, when creating entities in an entity-relationship diagram, the CASE tool could offer a list of similar entities or entities playing similar roles, *from which to choose. By building diagrams in this way,* from existing components, perhaps modified using subtyping, inheritance or other constructs, we go a long way towards avoiding the problems of Integrating new models with the existing repository. Here we touch on the general problem of reuse [12]. The concept of reuse is often associated with software, but it can also apply to models. For example, work on reuse by analogy has shown that it may also be possible to reuse patterns or common structures that occur in different situations.

The Visual Modeller prototype deals with this issue by providing the concept of a 'role'; whenever a component appears In a model, it does so In a particular role. For example, when a component of type 'person' is used in a model about car insurance, it could appear in the role of 'insurance claim assessor', 'customer', third party claimant' etc. Two levels of reuse are then enabled. first, the general requirements area (e.g. insurance' in the example given above' defines a set of components that can be reused in new models in the same or different requirements area.

Secondly, the type of each component (person' in the example above) provides access to the set of roles that the component might take. Obviously, if a given mole (or even type) has never appeared before in any model, then it has to be constructed afresh, but it can be based on an existing component or components.

Discussion

Hammer and Champy [2] show how business processes often continue with outmoded rules and assumptions from earlier times. The rules may well have been useful at one stage, but in changed circumstances, they become a threat to the effective functioning or the business. The purpose of business process re-engineering (BPR) is to identity these faulty ways of working and to change them. The redesign process inevitably involves significant change to the way people work; the process Is also often IT-led [13].

The usual reasons for carrying out BPR are cost reduction, time reduction, improvement in output quality and improvement in the quality of work life 113). why is BPR relevant to CASE tools and the systems analysis requirements gathering process? hammer and Champy Identify several conditions that am indicative of 'broken' those in need of redesign. Two of the symptoms they Identify are

o extension information exchange, data redundancy (and rekeying, when the process is computer-based): on the face of it, the use of CASE tools should not necessitate data redundancy. However, consider the earlier argument about the timing of CASE tool use. If the tool is used to document models that have *already* been formulated on paper, then there is indeed a problem of data redundancy. If most of the thinking' work Is done away from the tool, then the tool is not providing proper support.

o high ratio of checking arid control to value-adding: does CASE involve an inordinate amount of checking and control? Superficially, we might observe that CASE eases the checking task because it automates ft However, a deeper analysis would point out that, using the sort of techniques implemented in CASE tools, the need for checking is multiplied. As the techniques are separate and unintegrated, a checking problem is created in ensuring that parallel models are in step. For example, the single business fact that a customer has purchased a pair of shoes could well be represented in a CASE tool as one or more entities

in a data model, some processes on a data flow diagram (as well as data stores, data flows and their contents), and several states on an entity-life history diagram. All of these distinct representations of the same underlying 'fact' need to be reconciled with one another.

This brief analysis of Systems analysis and CASE in the light or ideas from BPR is not intended to be either conclusive or complete. It is intended to show that there is a *prima fade* case for rethinking the CASE-based systems analysis process, taking some first principles Into account Criticising the 'structured' methods Is not new, and the broad range of IS methodologies illustrates the divergence of opinion on what is necessary to ensure success in IS development [14]. There have been many comparisons of IS methodologies, often carried out in the hope of finding out what combination or features would make a wholly successfully development process [15]. Associated with this has been the development of 'portmanteau' methodologies, in which techniques, or indeed whole methodologies, are combined so as to obtain the benefits of many different approaches [16]. Although not decrying the potential of these ideas, perhaps more benefit would be obtained by simplifying and rationalising the development process, rather than Increasing its complexity still further.

This paper has presented some initial thoughts on ways in which redesign of the analysis process and its CASE support might happen. In summary1 these are

- by supporting an iterative approach to modelling, in which changing a model is as simple as visualising the change.

- by supporting less formal modes of modelling, if only at earlier stages of analysis, and allowing the refinement of less formal models into more formal ones.

- by providing a less IT-oriented user interface, hiding 'technical' details and procedures and carrying out house keeping activities without needing to be asked.

- by offering active support for a 'brainstorming'-style collaborative approach to modelling (in other words, being useable as the focus for group activity).

- by allowing an 'associative' approach to modelling, augmenting the experience of the analyst to permit the construction of models from existing components.

At this stage, these ideas are untested, and so evaluation of the prototype tool and technique by its target user population is an essential next step. Relatively controlled experiments should provide more Insight Into which features provide the most benefit, and why. It should be pos sible to find out if adjustrnent to the analysis process can lead to better understood requirements, better designs and, ultimately, products that meet user needs.

CONCLUSION

Computer-aided software engineering tools span every step in the software engineering process and those umbrella activities that are applied throughout the process. CASE comprises a set of building blocks that begin at the hardware and operating system software level and end with individual tools.

In this chapter we have considered a taxonomy of CASE tools. Categories encompass both management and technical activities and span most software application areas. Each category of tool has been considered as a point solution. In the next chapter, we consider ways in which individual tools are integrated to form an environment.

As the years pass, CASE will became part of the fabric of software engineering. Just as mechanical and electrical engineers rely on CAD/CAE/CIM for the analysis and design of high-technology products, software engineers will rely on CASE for the analysis, design and testing of computer-based systems for the twenty-first century.

During the software engineering process, sets of sequential tasks are coupled by a continuing flow of information. In addition, a set of "umbrella" activities occurs concurrently as one sequential task leads to the next. Each task and most activities can be assisted with the use of CASE tools. But the real benefit of the tools can not be realised until the tools are integrated – until other tools can easily use information produced with one tool.

The I-CASE environment combines integration mechanisms for data, tools, and human-computer interaction. Data integration can be achieved through the direct exchange of information, trough common file structures, by data sharing by interoperability, or through the use of a full I-CASE repository. Tools integration can be custom design by vendors who work together or can be achieved through management software provided as part of the repository. Human-computer integration is achieved through interface standards that are becoming increasingly common throughout the industry.

The application of three typical CASE products to the analysis and specification of requirements for manufacturing Information systems provided tangible 'hands on' familiarisation with CASE, and tested the appropriateness of the methods and tools employed.

All of the case studies attained the objectives set by senior management in an effective and structured manner. Indeed, the studies 'were considered as particularly successful by the organisations involved, and the impact of the methods and tools is most notable within the FOUNDATION case study, providing the basis and justification for revolution using the manufacturing organisation and control approaches.

We believe that the case studies demonstrate the relevance and considerable impact of software engineering methodologies and CASE tools on the specification of requirements for manufacturing Information systems. We consider Execrator to be the superior product of the CASE tools applied, and if coupled with methodology management and project planning and control facilities similar to those of FOUNDATION, it would provide a sufficient cornerstone for a Manufacturing CASE product.

The aim of the research presented here was to investigate the need for structured approaches and tools within manufacturing information Systems development, and the relevance of established software engineering methodologies and supporting CASE tools in meeting this need. The research topic has focused on what is widely seen as the most critical task within information systems development, the specification of requirements.

The first objective, the need for structured methods and tools in support of manufacturing information systems requirements specification, has been clearly established. An extensive review or the literature and a survey of manufacturing experts substantiated this need and the requirement for graphical modelling techniques. The success of the case studies within the collaborating manufacturing organisations in comparison with their traditional *ad hoc* approaches also confirmed that a disciplined approach with automated support was of benefit.

Commercially available software engineering methodologies and CASE products were examined for their suit-ability in providing support to manufacturing systems engineers In the task of requirements specification. This was the second objective of the research. To gain familiarity with CASE technology, the history and capabilities of both software engineering and CASE were established from the reviewed literature. A comprehensive survey of CASE products, commercially available within the UK, that support specification tasks strengthened this examination. Three leading products were selected from this survey and applied on-site to manufacturing information systems projects, to gain a

working familiarisation with CASE In a real and immediate situation to validate their suitability for the manufacturing systems engineer working within Industry.

To address the final objective, the main developments required for both CASE methods and tools and the sup-porting manufacturing Systems engineering organisation have been identified to facilitate the widespread application of CASE within this area. This has been undertaken on the basis of the literature review, use of existing CASE methods and tools, and views elicited from manufacturing experts and personal experience.

Here we present the conclusions drawn from the work undertaken and the main developments required if CASE is to find widespread adoption within the manufacturing industry. Substantial further research is outlined that will build on the results of this work

This paper presents the research undertaken in two traditionally separate areas, manufacturing systems engineering and information systems development The research has reinforced the need for structured approaches and tools in enabling manufacturing Systems engineers to analyse and specify information systems in support of manufacturing.

The relevance of software engineering methodologies and supporting CASE tools has been established in a more detailed manner than has previously been under-taken. The main developments identified in this work relating to the effective adoption of CASE within the manufacturing industry have been presented. These developments relate to the orientation of the methods towards manufacturing systems engineers, the ease of use of the supporting CASE tools, and the planned introduction of the methods and tools. Substantial further research has been presented, which Is aimed at extending the research work by defining the detailed requirements of manufacturing Systems engineers and appropriate CASE methods and tools In specifying manufacturing information Systems.

CASE tools should be enhanced so that full life-cycle activities are integrated with development activities. This points the way towards an architecture for CASE in the third generation.

In this paper, an architecture has been proposed for a 3GAWB where development tools are augmented by a super structure of

management control tools. Development tools should include editors for the techniques of structured methods and a word processor.

The main distinguishing characteristic of AWBs in the third generation is the primary interaction of users with the process model, rather than the development tools. The process model is used to guide developers in the steps of the method and control the use of the development tools in the environment. Process modelling should be based on accessible graphical techniques, and an enhanced data flow diagram/data dictionary notation is eminently suitable.

Another important characteristic of third generation tools is that they will be used not only by developers, but also by managers. Support should be provided for process modelling, project management and quality control. When these activities are supported In one environment, together with developmental activities, there is a cross-fertilisation of information that offers wider opportunities for automation. Although there are three distinct activities in the super-structure, or three logical strands, they should be implemented as a single physical strand (with a single user interface). Much of the functionality of this strand would be based on a single, unified task model. The user of this strand would be the project manager. This role is already responsible for project management and quality control, and would also carry out process modelling.

Early research results suggest that Implementation of the proposed architecture is achievable. It is particularly important that the superstructure could be added incremental to a second-generation CASE tool. This allows existing tools to migrate from the second to the third generation.

In the future, these ideas could be applied to other clusters of CASE tools relating to other phases of the development lifecycle. This could progress towards the integration of all types of CASE tools within a framework provided by a consistent and coherent management control superstructure, This vision of the future offers the best practical opportunity for realising the dream of an integrated project support environment.

Currently, tools in a CASE environment are invoked by developers based on process models that are in their heads, or at best in methodology or standards manuals. Within a third generation CASE environment the inter-action is directly with an explicit process model that provides guidance and control, in addition, the integrated automation of

management control activities is provided. This is imperative for coherent management of software projects. Such a CASE environment will be instrumental in the push for process improvement and improvements in software productivity and quality, towards the year 2000.

APPENDIX

Over the past 20 years, many SE have been many developing software for others. They have built complex systems that automate the work of others. They have used very little automation themselves. Infact until recently software engineering was fundamentally a manual activity in which tools were used only at the later stages of the process.

Today, software engineers have finally been given computer-aided software engineering tools (CASE), but they are not in as many varieties as we would like, don't provide enough sophistication and don't always match with what software developer use. But over time they will became more useable and more adaptable to the needs of individual practitions.

In this report, the technical aspect of computer-aided software engineering are discussed. Technologies as software engineering method and project management tools are explained. Tools and environments that will help to automate software technologies is also discussed.

PREFERENCES

1. JACKSON, M.A : System development (Prentice Hall international Inc. 1983)

2. JACKSON, M.A : Principles of program design (Academic press Inc. 1975)

3. GILB, T.: Principles of software engineering management, 1988

4. JONES, C.: applied software measurement: assuming productivity and quality, 1991

5. LOWRY, M. and DURAN, R.: Knowledge-based engineering, 1989

6. ROBINSON, K.: Putting the SE into CASE, 1992

7. PRESSMAN, R.S.: Software engineering: a practitioner's approach, 1992

8. LLOYD-WILLIAMS, M.: Intelligent assistance in the information systems design process, 1993

9. OXMAN, R. and GERO, J.S.: Using an expert system for design diagnosis and synthesis, 1987

10. GERO, J.S. and MAHER, M.L.: A future role of knowledge-based systems in the design process 1987

11. HULL, R. and KING, R.: Semantic database modelling: survey applications and research issues, 1987

12. HAREL, D.: biting the silver bulet, 1992

13. FUCHS, N.: Specifications are executable, 1992

14. TATE, G.: Prototyping : helping to build the right software, 1990

15. CIBORRA, C. and JELASSI, T.: Strategic information systems, 1994

REFERENCES

16. GUPTA, R. and E. HOROWITZ: Object oriented database with applications, 1991

17. HOPPER, M.D.: New way to compete on information, 1990

18. HORNER, M.: Future directions in CASE, 1990

19. PRESSMAN, R.S. and S.R. HERRON: Software shock, 1991

20. WASSERMAN, P.D.: Neural computing: theory and practice, 1989

21. FORTE, G.: In search of the integrated environment, 1989

22. FORTE, G.: Rally round the repository, 1989

23. MIKES, S.: X windows system technical reference, 1990

24. WELKE, R.J.: Meta systems on meta models, 1989

25. PCTE Functional specifications, 1988

26. WASSERMAN, I.A.: The architecture of the CASE environments, 1989

27. CASENEWS: Vol3, No 4, April 1989

28. LEE, E.: user-interface development tools, 1990.