Computer Science
and
Information Systems

Rekenaarwetenskap
en
Inligtingstelsels
Information Technology Research in the European Community

Pieter Kritzinger
Data Network Architectures Laboratory, Department of Computer Science, University of Cape Town
Private Bag, RONDEBOSCH, 7700 South Africa. e-mail: psk@cs.uct.ac.za

Abstract

It has been said that one reason why the US and Japanese information technology industries are ahead of those in Europe is that, where these countries see opportunities, European industry and its customers see primarily risks. Recognising this as well as the strategic importance of information technology and integrated communications systems for the future economic development of Europe, the Community launched the first of several five year programmes in information technology and public communication systems in the mid-eighties. In this report we describe the main programme called ESPRIT, how it works and some of the results achieved to date.

Introduction

The European Community consists of 12 nations with a GNP of US$ billion 4,862 (1988 figures). By the year 2000 the information technology and electronics sector of the European Community is likely to become the largest industry, representing some 300 billion ECUs (1 ECU = R3,82) or 6,7% of GDP. With the major impact these enabling technologies have on the competitiveness of the whole of a modern economy, Europe recognised very early that information technology is of crucial importance to the success of the planned, unified internal market and an essential factor in the Community's development strategy.

At the same time however, the positive balance of trade of the European Community in information technology amounting to some ECU 1,7 billion in 1975 was declining rapidly (it reached a deficit of almost ECU 22 billion at the end of 1988) and the Community decided something drastic had to be done. As a result it launched the First 5 year European Strategic Programme for Research and Development in Information Technology or ESPRIT I. It started on January 1st 1984 with a budget of 1,5 billion ECUs.

Table 1. The EC and its top 3 partners in numbers (1988)

<table>
<thead>
<tr>
<th>Population (millions)</th>
<th>GNP (US$ billion)</th>
<th>Per Capita GNP (US$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>West Germany</td>
<td>61,0</td>
<td>1 120,0</td>
</tr>
<tr>
<td>France</td>
<td>56,0</td>
<td>939,2</td>
</tr>
<tr>
<td>Italy</td>
<td>57,5</td>
<td>814,0</td>
</tr>
<tr>
<td>EC</td>
<td>325,1</td>
<td>4 475,1</td>
</tr>
<tr>
<td>USA</td>
<td>248,0</td>
<td>4 862,0</td>
</tr>
</tbody>
</table>

Source: US Department of State, Bureau of Public Affairs

The overall strategic goal of ESPRIT was to provide the European information technology industry with the technology base which it needs to become and stay competitive with the US and Japan in the 1990s. In addition to this primary objective, two secondary objectives were defined, namely:

- to promote cooperation in the information technology field between industries, universities and European research bodies on R&D projects up to pre-competitive level; i.e., prior to the development of commercial products, and
- to contribute to the development of international standards.

At about the same time the crucial importance of public digital telecommunications to the future social and economic infrastructure of Europe was recognised. Consequently a separate Research and Development programme in Advanced Communications Technologies in Europe (or RACE) was launched in 1985 with a budget of ECU 1,1 billion. The stated goal of this latter programme was

- to introduce Integrated Broadband Communication (IBC) into the European Community taking into account the evolving ISDN and national strategies while progressing towards Community-wide services by 1995.

Both these programmes have since progressed to second 5 year phases as, respectively, ESPRIT II with a budget of 3,2 billion ECUs, and RACE II with 1,039 billion ECU. In the meanwhile ESPRIT III is in its initial planning phases.

RACE is similar to ESPRIT in terms of its financing and organisation. Space does not allow us to detail all aspects of the programme in this report.

Strategic Themes

Although the ESPRIT programme broadly addresses the information technology and electronics industry, ESPRIT I had 5 major strategic themes.
1. Microelectronics. This field was perceived as the key strategic area for information technology R&D in the future.

2. Software Technology. The stated goal of this research area was to do what was necessary to put the software development process on a sound engineering footing. Sub-areas were defined to deal with formal methods, development tools, management aspects, quality measurement and the development environment.

3. Advanced Information Processing. This area covered knowledge-based systems, new computer architectures and speech- and image-processing.

4. Office Systems. When initially conceived in 1984, this application area was viewed as of strategic importance for the efficiency of business throughout the Community.

5. Computer Integrated Manufacturing. This area comprised the total range of computer integrated manufacturing activities, including: computer aided design (CAD), computer aided engineering (CAE), computer aided manufacturing (CAM), flexible machining and assembly systems, robotics, testing and quality control. The area was selected for its potential impact on the methods and economies of production, particularly in the information technology industries, and also for the manufacturing industry in general.

In addition, the Information Exchange System project was started with the twofold objective of:
- providing communication services to ESPRIT participants, both industrial and academic; and
- encouraging the development and adoption of OSI standards.

It is indicative of the experience gained in ESPRIT I and technology developments since it was started, to note how the strategic fields chosen for ESPRIT II differ from those of ESPRIT I. R&D in ESPRIT II is carried out in the following four major areas:

1. Microelectronics was retained as the key strategic area for information technology R&D in the future.

2. Information Processing Systems and Software. The work in this field will provide the fundamental and generic technologies which will support the development of information technology products expected on the market in the next decade. Thereby ESPRIT II recognised that information and its efficient use is not only a means of administration and communication, but that it is part of an enterprise's competitive advantage.

As an aside, it is interesting to note that, of the 30 billion ECU expenditure on software and services in 1989, about 50% was provided by the manufacturing, banking and other financial services. This is expected to remain true through to 1994, when the market is expected to be worth 70 billion ECU. About one third of this market comprises customer services, consultancy, training and services while packaged software represents about 40% of the market. The latter component is expected to increase to 50% of the market by 1994 with services and training remaining constant at 30%.

3. Advanced Business and Home Systems and Peripherals. It is clear that information technology in the business environment is moving to advanced integrated systems capable of serving all the functions of the enterprise in an integrated multimedia environment. The priorities for work in the Community documents reflect these salient points.

4. Computer Integrated Manufacturing. The emphasis in this strategic area has not changed significantly from ESPRIT I to II.

In addition to the above, the Open Microprocessor systems Initiative (OMI) was started in ESPRIT II. The major motivating factor for the Community was the 82% dependence on non-European sources for microcomponents, representing 7 billion ECU in 1989 and which is expected to rise to 16 billion ECU by 1994.

**Funding**

ESPRIT is an industrial programme and it was not started for, or by, academics. The main driving force behind the ESPRIT I programme was industry, who first defined the research areas and then the goals and workplans. Industry was represented by the largest 12 information technology companies (known collectively as "The Twelve") in Europe.

ESPRIT R&D projects are implemented by shared-cost research and technological development contracts, with the Community financial participation normally not exceeding 50%. Universities and other research centres participating in shared-cost projects have the option of requesting, for each project, either 50% funding of total expenditure or 100% funding of the additional marginal costs. ESPRIT projects have a maximum duration of 5 years but should normally be shorter.

In the case of ESPRIT I, the Twelve received 50% of the ESPRIT budget and were involved in 70% of all projects. Small- to medium-sized enterprises (SMEs) participated in 65% of the projects and received 14% of the funding. The funding allocation by sector participating in ESPRIT I is illustrated by the chart in Figure 1.
Basic Research

While ESPRIT I made no special provision for basic research, ESPRIT II includes a sub-programme, with a budget of 130 million ECU, aimed at developing new knowledge and expertise in the basic disciplines considered essential to secure the long-term future of information technology in Europe. Some 62 projects have been selected to carry out basic research in areas such as super-conductivity, optical and neural computers, speech and image processing and so on. In all, 211 university laboratories, 57 research bodies and 17 industrial companies are participating in these projects.

Apart from such projects, basic research activities also involve:

- Working Groups which are concerted efforts to improve the systematic exchange of information and for which short scientific visits and workshop are funded, or
- Networks of Excellence which are composed of both academic and industrial teams geographically distributed throughout the Community. These are set up to provide a critical mass of complementary knowledge and expertise and to share limited and expensive resources. Funding for Networks of Excellence is restricted to the marginal costs of establishing the administrative and communications infrastructure necessary to carry out the coordination.

The evaluation criteria for basic research projects are less specific about the value for market exploitation of the expected results and more specific about conformity with the basic ESPRIT technical objectives, inter-disciplinary nature and scientific calibre of the partners.

Programme Management

Participation in the programme is solicited by a "call for proposals" made by a "consortium" comprising at least two participants or "partners" from different members within the Community¹ and usually no more than six – except for standards projects.

The proposals are then evaluated by external experts who take account of the following points in particular:

- The impact and potential for industrial exploitation of the expected results of the project.
- Eligibility of the partners.
- Technical merit of the proposal including a justification of the proposed theories and methods.
- Soundness of the proposal with regard to issues like the assessment of major technical risks and technological advances expected.
- All proposals are scrutinized for human and organisational factors to ensure that the results would be appropriate for the intended user base.
- Soundness of project plans with respect to the distribution of effort, clear and well defined roles for each participant, realistic timescales and the proposed management structure and methods of supervision.

Once a project has proceeded to contract signature² it is periodically subjected to four different audits until its completion:

- A strategic audit is carried out periodically to examine the evolution of the political, economic and social objectives in the light of world-wide strategic developments.
- An annual technical audit examines the progress of all projects which comprise the Programme. It is performed by a team of independent experts.
- A programme management audit evaluates overall management performance as well as individual project management and deliverables.
- The usual financial audit is done to ensure the correct use of public money.

Results

A total of 227 projects were implemented during ESPRIT I. They involved 536 participating entities and some 3000 full-time researchers.

- Of the 327 participating industrial companies, almost 45% were firms employing fewer than 500 people and 40% of those employed fewer than 50. SMEs were extremely active, being involved in more than half the projects and being responsible for more than 25% of the research work in 60% of the cases.
- Nearly 200 universities and research institutes participated in approximately 70% of the projects. In more than half the cases, these scientific institutions were responsible for at least 25% of the work.

Towards the end of ESPRIT I, nearly 165 projects had delivered concrete results. Of those, 75 had already helped to put specific products and services onto the market, while for another 60 projects, the research worked had resulted in the transfer of technology for uses not directly linked to the project itself.

One detailed example is work in the Information Processing Systems and Software sub-field which let to the definition of a reference model for CASE (computer-aided software engineering) tools that has been adopted by the European Computer Manufacturing Association (ECMA). This has let to requests from the US National Institute of Standards and Technology to collaborate on the ECMA model as the basis for their own work on a reference model. Details about this and all other European Community research projects can be obtained from CORDIS mentioned below.

ESPRIT I participants who were questioned about their perceived successes of the programme considered increased knowledge as the most important benefit (69%), followed by a belief that research goals more ambitious than would otherwise have been set, had been reached.

¹Partners in ESPRIT from outside the community are not eligible for financial support. A programme called EUREKA fosters extra-European research.
²Only 20% of all proposed projects in the case of ESPRIT I.
There have been direct benefits in being able to cover a wider range of research topics quicker by sharing results with the project partners. A significant number of responses claimed a contribution either to existing products (35%) or new products (45%). It was felt, however, that there needs to be a greater degree of concerted action by project teams and a sharper strategic focus on market opportunities while, simultaneously, basic research must continue and even be increased. 15% saw no direct benefit.

Apart from technological reasons, ESPRIT and RACE were started, in the first instance, as Community programmes to promote cooperation in the information technology field between industries, universities and European research bodies on R&D projects. The extent to which this was achieved is thus an important criterion for measuring its technological successes. In this respect it is a general consensus that ESPRIT has indeed achieved a profound change in attitude in the Community. Cooperative, pre-competitive research and development is now a formula which is working effectively.

**Summary**

It is apparent from the many ESPRIT reports that some participants in ESPRIT I, particular those from the Twelve, were originally rather sceptical about the likely successes of the programme. No small reason for this was that they had no accord on the product priorities for the industry as a whole.

Five years ago, the largest European companies viewed one another much more as competitors than collaborators. Five years later, however, apart from the major technological progress, a major, if not the major achievement is that there now exists a spirit of pre-competitive cooperation in the Community to the common advantage of all.

**Further Information**

The European Community has set up an on-line information service to give quick and easy access to information on European Community research programmes. The Community Research and Development Information Service (CORDIS) is at present offered free of charge and comprises eight databases.

More information and CORDIS registration forms can be obtained from

**ECHO Customer Service**
**CORDIS Operations**
BP 2373
L-1023 Luxembourg
Tel.: (+352)34 98 11 Fax.: (+352) 34 98 12 34

Pieter Kritzinger is professor and currently head of the Computer Science Department at the University of Cape Town. During 1992 he was on research and study leave at the University of Dortmund in Germany and was thus able to observe programmes like ESPRIT and RACE at close hand.

---

**Editor's Notes**

A number of the articles in this issue of the South African Computer journal are in the field of Information Systems. Research in this area is beginning to blossom in the country. There are probably many more researchers in Information Systems than there are in Computer Science. It is hoped that not only academics, but also professional practitioners will submit articles.

Research in this area normally falls into three main categories. The first of these is pure research. This is a difficult area. Few researchers make a contribution here, mostly because the theory progresses slowly. However, these articles are to be encouraged. The second category of research is the collection of information from a variety of people in the field by means of questionnaire or interview and the use of this data to formulate policy and trends. An important aspect of this research is in order to corroborate theories or to identify areas where new theories are needed, or old theories amended. This has proved to be a very fruitful area of research and many beneficial results have accrued from it. The third category of research is performing careful analysis on a specific Case Study. In this area the case under study will need to display something which is innovative, either in the system itself, or in the way it was implemented. The case will need to prove something new and important or to break grounds into areas which have not formally been addressed.

All three types of work is worthy of publication if the results that they deliver are of benefit to the community which they serve. All three will be considered for publication by this Journal.

The journal divides into two sections. The primary section is involved with research while there also is a section on viewpoints and communications. Articles submitted for the latter are not refereed, but can be included after study by the editors. On some occasions articles submitted for the research section have been found appropriate for this section. This policy also applies to articles in the Information Systems field.

**John Shochot**
Subeditor: Information Systems

---

This issue was produced with the kind support of Software Collage (Pty) Ltd.
Abstract

Software Engineering is a difficult and time-consuming discipline to learn. CASE Tools are generally expensive, prescriptive and tedious to use. Structured Software Development Environment (SSDE) was developed as a teaching aid which can assist students with aspects of the software development life cycle, with a minimum of cost and effort. SSDE provides the system designer with an interactive menu-driven environment, and a framework within which he can conveniently express and manipulate his proposed solution. This representation is in terms of both a conceptual model and a detailed software logic definition. SSDE provides tools for both high-level (or logical) and low-level (or physical) design. It allows a user to follow his own preferred methodology rather than restricting him to one specific strategy. Experimental usage of SSDE at the University of the Western Cape showed that it is an effective tool for students of Software Engineering.

Keywords: Case tools, software engineering education, structured development.

1 Motivation

SSDE Objectives
The aim of SSDE is to provide an automated environment for software engineering students. As such it must be inexpensive, easy to learn and use, and flexible. It must cover high-level and low-level design and be targeted at the personal computer running DOS. Naturally it can also be a valuable aid in solving real-life problems.

SSDE is directed at personal computer users, rather than the user of a large or medium-sized machine because of the cost and availability factors. A secondary reason for concentrating on the small user is because of the declining popularity of mainframes which is being spurred on by the tendency to rewrite or revise mainframe software for micros (eg. SPSS/PC and SAS/PC) [18].

A major design goal of SSDE was to encourage structured and modular techniques. It is well known that such techniques offer many advantages over adhoc methods [13, 10, 8, 15].

SSDE and Diagramming Techniques
Diagramming can be considered a vehicle for clear, concise and structured design and it is for these reasons that SSDE uses diagramming techniques.

The high-level diagramming techniques used here are lacking in constructs to show low-level design logic. It was for this reason that the high-level diagrams were combined with a diagramming technique which is well suited for low-level design. A low-level diagramming technique has all the necessary constructs to effectively design structured detail logic.

Complex system and program logic can be made much easier to understand with a good, clear diagramming technique. Diagrams advance team communication and enable management to implement and monitor control. Diagrams are also language independent [7] and provide a design consistency [17].

Outputs
Writing code for designed modules is a time consuming task when done manually. A prescriptive low-level design methodology is used, allowing for SSDE to generate skeleton programs. Incomplete text or abstract entries are generated as comments.

SSDE provides documentation for the high-level hierarchical design and for the low-level detail design. This documentation can eventually contribute towards the contents of the project information file. Besides the obvious advantages to the student designer and the teacher, such reports are useful for teamwork exercises. A skeleton User Manual is also produced.

Software Solutions
Most end-users are not literate in program and application writing, but are able to solve problems in their particular domain [4]. This should encourage the development of tools which will enable users to construct their own solutions. Furthermore, the trend to-day is to involve the end-user much more in the design phases of a system, and the use of language-independent diagrams will greatly encourage the users’ contribution [19, 16, 9, 5].

2 Overview

An Automated Design Environment
Structured Software Development Environment (SSDE)
Structured Software Development Environment (SSDE) provides the system or program designer with a number
of automated tools which allows him to express his software solutions in an orderly and organized manner. These tools are all integrated so that the designer can easily move within the environment from one facility (tool) to the next. The analyst interacts with the environment via a menu system and the interface is designed for fast and easy usage. The major facilities provide support for modular decomposition, data structure design, algorithm construction and documentation production. In addition, the system's software engineering databases can be queried and skeleton code can be generated automatically.

High-Level Design

The environment allows a high-level functional design to be performed by means of a hybrid HIPO-Structure Chart diagramming technique[12, 21]. The high-level design is an overview of the complete system being constructed. Using the HIPO-Structure chart combination the system can be represented as a hierarchy of modules.

Thus system modules and their inter-relationships are shown, not the actual low-level logic. Each functional component ("box") in the visual representation can represent a system, subsystem, program or program module. Inside the "box" the designer can enter text to give information about that particular component. Not much detailed design is required here, since the object of the design phase at this point is to show the major components and their relationships with each other.

Designing a system involves the process of decomposition. This process is very much an informal procedure with the analyst assigning names to the various modules in a natural language [1]. During the decomposition phase, the analyst specifies the functional components (modules) and further also specifies the input and output variables of each. This High-Level design facility may also be used to annotate and design a data structure.

Low-Level Design

The Low level design is done using an extended form of the Chapin [6] and Nassi-Shneiderman [14] diagramming technique. The extension provides for abstraction and supports the use of secondary diagrams in order to build larger programs. The analyst can design his detail logic using this technique. He can edit or re-structure his design as it grows and changes (incremental design is supported). Secondary diagrams allow the process of stepwise refinement and modular design to be used by the system architect. Secondary diagrams are used when the detail logic of a certain specific step in the main (or primary) diagram occurs as another (secondary) diagram. This feature allows the designer to avoid large diagrams. It can also be used to postpone the consideration of some specific logic while designing the overall logic of the primary diagram.

An abstract entry is also permitted if the designer is unable to show exactly how a step is achieved, but only states what it does, using natural language. Enough precise information is captured about the low-level design in order that skeleton code can be automatically generated by SSDE. Pascal programs are currently produced, as this is still a common teaching language. The more rigid (less abstract) the requirements for low-level design information, the easier it becomes to generate skeleton code.

Software Engineering Databases

A central repository, consisting of a number of software engineering databases, is maintained. The databases can be independently queried to determine the systems being developed and/or the details of specific designs. The database subsystem and the historical information they contain enable teachers to rapidly discover students’ progress.

Validation Mechanisms

Numerous validation mechanisms exist throughout the environment to ensure, wherever possible, that the designer is proceeding with his design in a meaningful manner. For example, within the low-level design constructs (Loop, Conditional etc), the text is scanned to guarantee that it is appropriate for that construct.

Methodology

The analyst can choose to follow his particular methodology or a combination of methodologies. In this way SSDE can be used to experiment and experience several different software engineering approaches.

Auxiliary Facilities

System documentation for both the high-level and low-level detail design is provided. As such it is valuable during the design phase for teams working together, because a history of the design progress, an up to date record, etc, is provided. SSDE creates a skeleton user manual which the designer can edit. Previously designed systems can quickly be retrieved, modified and re-used. It is possible to design a common activity only once and subsequently to re-use this particular logic elsewhere in the same or other systems. This can be done with both the high-level and the low-level diagrams. Thus the reuse of existing software in the construction of new applications is made possible [11].

Components can also be re-used within the same system. Besides illustrating the advantages of reusability, this feature also permits incremental development of large systems for novice software engineers.

Programming Information

SSDE was developed and written in TURBO PASCAL version 3.0 [3]. This was chosen because of its development environment, and the use of Pascal makes it possible for a wide range of students to experiment with, and extend, the SSDE system itself.

The Database components were implemented using the TURBO PASCAL Database Toolbox [2] which uses B+ Trees.

Summary

SSDE is an interactive, menu-driven development environment which permits high- and low-level system and program definition. It includes a central database repository,
design documentation and skeleton code generation in addition to facilities for manipulating, verifying and re-using design diagrams. SSDE was developed using TURBO-PASCAL with the TURBO-PASCAL DATABASE TOOLBOX.

3 System Architecture

The overall system architecture is shown in Figure 1. This is reflected in the organisation of the SSDE menus. Following a brief look at the user interface, each of the major sections – design, query and document-generation – is outlined here.

The SSDE screen is laid out in the same way as is done for most PC software: with a menu bar, a status panel, a message area, etc. The Low-Level design is presented by drawing the appropriate “box” on the screen as each construct is selected, so that a Nassi-Schneiderman diagram is displayed. An example is shown in Figure 3. Scrolling occurs automatically in large diagrams, and it is possible to jump forward or back to particular parts of such designs, if necessary. The user is able to easily position the cursor at the appropriate menu option, in order to edit existing text, or insert additional material. Editing a Low-Level design causes the diagram to be automatically adjusted when necessary. The ease and speed with which a pseudocode solution can be built and adjusted makes this an interesting and useful tool for any application developer.

The display of the High-Level design is complicated because diagrams can grow in two directions, i.e. both downwards and across. Thus it is not practical to show too much information for any one component, as this would require excessive scrolling and make it difficult to see the overall picture without recourse to separate sub-diagrams. On the other hand, the information should not be too cryptic. As a compromise, a shortened character name is used to identify each module in the diagram, and boxes are avoided as they require additional columns and provide no extra information. The IPO (Input Process Output) specifications for a module are displayed on a separate screen, so as not to clutter the top-level view. Naturally all information on modules is obtainable through SSDE documentation and querying facilities. Once again, scrolling and cursor positioning are made as simple as possible through the use of arrow keys. An example screen can be seen in Figure 2.

The Design Subsystem handles development work, permitting creation and alteration of High-Level and Low-Level solutions, as the data organisation, program structure and pseudocode evolve. Automated skeleton code generation is also invoked from here. Additional facilities are provided for checking and analysing the current designs: for example, a Verify option indicates variables that were given as Input or Output of a module (in the IPO specification) but are not used in the corresponding Low-Level module. Cross reference lists and information on pending designs can also be obtained; the latter indicates places in the system where further work is required, e.g. a High-Level module of which the Low-Level component is outstanding.

The Database Subsystem allows the user to browse through any number of designs, including historical data. Specific kinds of information can be extracted, such as all past designs of a particular system, all abstract entries, all variables referenced, etc. (See the section The Low-Level Database Sub-Menu later.)

The documentation Subsystem presents a menu of the various kinds of reports that SSDE can generate, enabling appropriate material to be collected according to the targeted audience and envisaged usage. The options are illustrated in the section The Documentation Sub-Menu.

The Overall Menu System

The diagram in Figure 1 provides an overview of the various menus available and their interaction.

The Main Menu

The user is presented with the three major options (facilities) provided by SSDE from which he can make his choice depending on what activity he would like to do next. The main menu presented looks as follows:

```
This menu permits the choice between the three major system activities and help screens.
```

The Design Facility Sub-Menu

The following sub-menu is presented when the design option has been chosen from the main menu:

```
Option: High-Level Design and Low-Level Design. Figures 2 and 3 show high-level and low-level design screens.
```

```
Option: Verify HL – LL Link. As part of the definition of a high-level component, variables are defined which constitute the input/output specifications for such a component. If such a high-level component has an associated low-level design, then these variables will also appear in the low-level component.
```

```
Option: Generate LL Code
```

This option is selected when skeleton code for a particular low-level design is required.
**Figure 1. The Menu System**

**Figure 2. Global View and Menu**

**Figure 3. The Low-Level Design and Menu Screen**
The Database Query/Update Sub-Menu
This menu is displayed when the database option has been chosen from the main menu:
1. High-Level Systems
2. Low-Level Systems
3. Return - Main Menu
H=help

The High-Level Database Sub-Menu Options
When "High-Level Systems" from the above menu is chosen the following menu appears:
1. Save
2. List
3. Select
4. Delete
5. Find Variable
6. HL Components with no LL
7. List Deletions
8. Return - Main Menu
H=help

- Option: Save, List, Select, Delete.
  These are the normal database functions.
- Option: Find Variable.
  All the components can be identified which have referenced a specified variable, whether as input or output.
- Option: HL Components with no LL.
  List all high-level designs which have no dependents and for which no corresponding low-level design has been completed.
- Option: List Deletions.
  Lists previously deleted systems so that they may be recovered. To recover any deleted system, it can be retrieved and then renamed.

The Low-Level Database Sub-Menu
1. Save
2. List
3. Select
4. Delete
5. List Variables
6. List Abstractions
7. List Deletions
8. Return - Main Menu
H=help

- Option: Save, List, Select, Delete.
  These are analogous to their high-level counterparts.
- Option: List Variables.
  List all variables, their types and structure, for a particular low-level design.
- Option: List Abstractions.
  Locate all abstract entries in a particular low-level design.

The Documentation Sub-Menu
This menu is displayed when the documentation option is chosen from the main menu:

PRINTER OPTIONS
1. Print Current HL Design
2. Print Current LL Design
3. Print all LL Designs
5. Print Cross-Reference
6. Return - Main Menu
H=help

- Option: Print HL Design and Print LL Design(s).
  This prints out a hardcopy of the high-level (or low-level) design together with all associated information.
- Option: Generate Manual.
  This will generate a skeleton system manual write-up from the high-level definition of the system.
- Option: Print Cross-Reference.
  A particular variable may be entered and all high-level functional components which have a reference to that variable are listed.

A Typical Development Process
Although different designers will use varying design approaches, the following steps give a typical example of a design strategy:

- Define a data structure using the high-level hierarchy. Since structure charts are a form of decomposition, the data structure can be represented in terms of a hierarchy. The inter-relationships between the elements can be illustrated and use can be made of the associated IPO view for details.
- Create the functional components using the high-level design. A fast design can be done specifying only the names of the components. The detail (e.g. variables, input/output, process steps and general comments) of each component can be added later.
- When difficulty is encountered during the high-level design, then the process steps for that component can be stated or a low-level design done with abstract entries. The clarity obtained in this way will facilitate further progress.
- Low-level designs for functional components can be entered.
- Print the low-level and symbol data.
- Verify high- to low-level linkages.
- Generate skeleton code.
- Complete code where necessary.
- Test code.
- Print final documentation, manual and cross-references.
- The database query facility can be used during maintenance to find the relevant part. Low-level and high-level designs can be copied and changes made.

Summary
SSDE provides the designer with a functionalized menu-system which distinctly groups all similar activities together.
SSDE: Global View
System=financial-system

payroll
| calcnetpy
| calcgoss
| normlpy
| taxded
| payetx
| sitetx

Low Level Designs: normlpy overtm payetx

Components which have been designed:

- **payroll** Module name=payroll calculation
  Input=emp.record: pay_rec
  Output=gross_sal: deduct: net_sal
  Comment=payroll calculation for salried staff
  process=get employee record
  calc gross and net salary

- **personnel** Module name=personnel system
  Input=emp.number: update_rec
  Output=emp.record
  Comment=Update personnel records
  process=get employee number & update detail
  print out new employee record

Similar prints for the other modules on the Global View

Figure 4. High-Level Printout Documentation

System=inventory-reorder
Component=inventory-reorder-add-invoice

open database file "invoice.dat"
opent index file "invoice.ndx"
get invoice-type%
until invoice-type% = 1 or 2 or 3
**update-database

Figure 5. Low-Level Printout Documentation
4 SSDE in Education

The sub-systems within SSDE can be used together or separately to assist in the teaching of several components of a typical Computer Science course. In addition to using the tool as an aid to managing the software life cycle as a whole, the High-Level Design facility can be required in all practical work, to ensure that assignments are tackled in a top-down, modular fashion. This component can also be used in the study of data structures and database design, to represent complex data organisations (e.g. inheritance hierarchies and objects in an object-oriented environment), to derive relational database schemes using decomposition techniques, etc. In addition, the concept of Software Engineering Databases is more readily understood when an actual, and sufficiently simple, example is available. The Low-Level Design system can also be useful in its own right for developing smaller programs and languages.

The SSDE tools are suited to teaching many different facets of software development: working with partial solutions, project development as a team, software re-use, maintaining systems developed by others, etc. The availability of a cheap, automated system opens up new possibilities for the teacher. He can set up partial systems and ask students to determine the requirements from the SE databases, complete different sections accordingly, etc. Experimentation with practical exercises is possible. For example, different groups of students can be given separate systems to design. Individual groups can change to new systems for different development stages: eg. a group might design system A then develop low-level modules for system B. The interactive and document generating facilities ease the task of subsequently evaluating student performance. As SSDE is written in Turbo-Pascal, it is also possible to arrange projects for extending the environment itself.

5 Evaluation

Introduction

The contribution that SSDE can make towards supporting the software development life cycle is investigated and other advantages offered are mentioned. SSDE was evaluated by a number of users and the results of this exercise is reported.

SSDE – Its Role in the Software Development Life Cycle Planning, in the sense of an initial investigation, is possible via the high-level design facility. The designer can quickly use the environment to perform an initial outline of the system for presentation to other interested parties and for use in a feasibility study.

The environment provides adequate support for this stage through the high-level design facility. Using this the designer can view, edit or print the complete conceptual model at any point during the design [20]. Physical design is concerned with the more detailed design of the system. Through the use of the low-level design facility the analyst can design his low-level logic in as much detail as he requires. This logic can at any point be revised and/or re-structured. Design refinement, in terms of using secondary low-level designs, is also supported.

A complete document of the system design can be produced which can greatly assist the programming staff in the actual implementation of the system. Together with skeleton program production, these facilities should make a considerable contribution towards improving implementation schedules.

By viewing the system via the SSDE Database Query Facility or its documented output, maintenance personnel can quickly familiarize themselves with the overall system.

Advantages Offered by SSDE

An inherent design methodology is present which guides the system architect through a design cycle starting at the high-level (broad outline) of the system and proceeding to a low-level (detail) logic design. Beyond this, SSDE aims at flexibility, so the designer can follow his own methodology. Incremental system design is possible. It is possible to design, implement and run a partial system and then the system can “grow” as new components are added. The reuse of existing software in the construction of new applications is encouraged.

Ease of communication is made possible by the graphical facilities and the documentation provided. Graphical representation clarifies a user's perception of a system and makes it easier to commit to a particular design option or not. Complex systems can be simplified by abstraction. Increased productivity is provided by the automation of design components, e.g. diagram production, documentation, editing at all levels, skeleton code generation, etc. Maintenance should be facilitated due to the consistency of systems produced with SSDE.

Menus allow for easy diagram manipulation and movement within the system. They are simple enough for even novices to use, yet allow fast navigation for experts. The manipulation of program constructs in a language independent manner is supported. SSDE and any designs produced by it are completely portable and can with minimum effort be transferred to any PC running DOS and no graphics adaptors or mice are required. For the user of a fixed disk, there is no noticeable delay when secondary storage is used. The drawing of high-level and low-level diagrams is done quickly and the designer is not delayed in any way.

User Survey

In order to measure the usefulness of an environment such as SSDE, an evaluation exercise was conducted. A group of senior undergraduate computer science students were presented with a typical application problem and a questionnaire to complete.

The evaluators were required to design a financial information system which included payroll, credit card application, economic indicator reporting and banking systems. To ensure the evaluators gained as much experience as possible with every facet of SSDE, they were assigned
a large number of design tasks, not all of which had to be completed in full. They were required to complete all high-level designs. For the low-level detailed definition however, they were only required to complete some components. This latter constraint reduced the design undertaking and made it possible for the evaluators to experiment with the many other facilities provided by SSDE.

The response to SSDE was overwhelmingly positive. Its success was confirmed by the absence of negative feedback. The automated design support with editing at the high- and low-level design stages, the database facilities and the code generation capability were well received. Students mentioned that SSDE demonstrated "potential" as a software development environment and that it was easy to learn.

The histogram in Figure 6 summarizes the responses to some of the more important questions answered by the evaluators. It can be seen from this histogram that the responses were very positive in general.

From a teaching point of view the following improvements are experienced when compared with the non-automated design methodology:

- Design information is always neatly presented.
- It is easier to mark designs because of the neatness and structure.
- Larger design tasks can now be completed in a shorter period of time.
- Automated revision (corrections, additions, deletions and enhancements) is possible.
- Software re-use and rapid-prototyping (with their associated advantages) can now be demonstrated and experienced.

Summary

SSDE provides automated support for the different phases of the software development life-cycle. Facilities such as on-line databases, pictorial presentations and code generation help the analyst to create a conceptual model, to define the functional components which constitute the proposed solution, to spell out the detail logic required to accomplish these, and to compare and analyze alternative designs.

6 Conclusion

This research aimed at providing an automated facility which strives to reduce the number of manual tasks the systems designer has to perform, and which is cheap and easy to use.

The evaluation of SSDE showed that students of software design prefer an automated tool or environment above the normal manual methods. Design automation results in systems being designed much quicker than would otherwise be the case, increasing the designer's productivity.

The evaluation exercise clearly showed that the initial goals set for SSDE were accomplished. Several insights, as regards design automation, were gained as a result of the research. Scope for future enhancements exist but these must not be incorporated if it is at the expense of SSDE's ease of use, flexibility, low cost and efficient performance.

The system has potential for extension in two major directions. On the one hand, the Software Engineering features can be expanded so that, for example, code is generated in other languages and more attention is given to data structure design (e.g. Entity-Relationship diagrams, automatic relation scheme generation, etc). In addition,
education aids for both student and teacher can be incorporated, providing for self study, design evaluation, progress monitoring, etc.

References

12. IBM. HIPO - A Design Aid and Documentation Technique, May 1975.
Contents

GUEST CONTRIBUTION

Information Technology Research in the European Community
PS Kritzinger ................................................................. i

Editor's Notes ................................................................. iv

RESEARCH ARTICLES

Assessing the Usability of Computer Systems
TC Parker-Nance and PR Warren ........................................ 1

An Enquiry Into Property-Goal Type Definitions Of The Term “Information System”
M Mullany, A Dos Santos Gomes and R Miller .......................... 8

The Key Issues in Information Management for the Mid-1990s. Back to business basics through the commercialisation of the ISD
DSJ Remenyi ................................................................. 14

SSDE: CASE in Education
MJ Norman and S Berman .................................................. 23

Die Versoening van Konflikterende Doelwitte in die Databasisomgewing met behulp van Regressiemetodes
DB Jordaan, JM Hattingh en T Steyn .................................. 32

A Simulation Environment for Evaluating Distributed Query Processing Algorithms
SM Lamprecht ................................................................. 37

Die Empiriese Evaluering van Variasies van Woordpassingsalgoritmes vir die Bepaling van Ekwivalente Vanne in 'n Genealogiese Databasis
GdeV de Kock en C du Plessis ........................................... 48

Expert Systems for Management Control: a Multi-Expert Architecture
V Ram ............................................................................. 54

COMMUNICATIONS AND REPORTS

Artificial Intelligence, Expert Systems and Business Computing: Is There a Problem?
TD Crossman ................................................................. 60