OCCASION

This publication has been made available to the public on the occasion of the 50th anniversary of the United Nations Industrial Development Organisation.

DISCLAIMER

This document has been produced without formal United Nations editing. The designations employed and the presentation of the material in this document do not imply the expression of any opinion whatsoever on the part of the Secretariat of the United Nations Industrial Development Organization (UNIDO) concerning the legal status of any country, territory, city or area or of its authorities, or concerning the delimitation of its frontiers or boundaries, or its economic system or degree of development. Designations such as "developed", "industrialized" and "developing" are intended for statistical convenience and do not necessarily express a judgment about the stage reached by a particular country or area in the development process. Mention of firm names or commercial products does not constitute an endorsement by UNIDO.

FAIR USE POLICY

Any part of this publication may be quoted and referenced for educational and research purposes without additional permission from UNIDO. However, those who make use of quoting and referencing this publication are requested to follow the Fair Use Policy of giving due credit to UNIDO.

CONTACT

Please contact publications@unido.org for further information concerning UNIDO publications.

For more information about UNIDO, please visit us at www.unido.org

UNITED NATIONS INDUSTRIAL DEVELOPMENT ORGANIZATION
Vienna International Centre, P.O. Box 300, 1400 Vienna, Austria
Tel: (+43-1) 26026-0 · www.unido.org · unido@unido.org

RESTRICTED

DP/ID/SER.A/706
10 June 1986
ENGLISH

15627

ADVANCED MANUFACTURING AND ENGINEERING METHODS

DP/BUL/81/009

BULGARIA

Technical report: Computer aided design (Part II)*

Prepared for the Government of Bulgaria
by the United Nations Industrial Development Organization
acting as executing agency for the United Nations Development Programme

Based on the work of M. Vandersluis
Expert in computer aided ~~design~~

United Nations Industrial Development Organization
Vienna

---

* This document has been reproduced without formal editing.

V.86 57226

CONTENTS

1. Introduction

This report describes my two week mission to Bulgaria for the UNIDO project DP/BUL/81/009, between 22/2/86 and 8/3/86, and consists of observations made during the mission together with recommendations where appropriate. The mission was a continuation of a previous visit by Senior CAD Advisor Dr. Keith Shaw, also from PAFEC Ltd (ref.1). The major part of the mission consisted of training of various forms, which is described more fully in the following section. The major conclusions of the mission are then summarised.

2. Training

During the period of the assignment, extensive training was given in accordance with the Job Description DP/BUL/81/009/11-02/31.9.E, and taking account of the requests made by the CAD Laboratory in their telex no. 025/17.02.1986, a copy of which was sent to UNIDO.

2.1 Project Training

Training was given to approximately 30 members of the CAD Laboratory in the form of lectures followed by question periods. The lectures covered many systems aspects concerning development, implementation and use of CAD systems. Each subject is described in more detail below. A final Question Period was also held, where Project Members could raise points concerning any aspect of CAD system development.

2.1.1 Introduction to CAD Products

The various components of a typical modern CAD system were discussed. Each component or 'module' may be linked to others, with data communicated between the modules by using a common data structure or an intermediate file. Typical modules were described, with examples of the graphical output which may be produced by each module. The modules described included

2.1.1.1 General 2D Module

This is the basis of most CAD systems and contains many general purpose options (e.g. multiple copy, drag, delete area) which may be utilised in order to create drawings quickly and efficiently. Other facilities normally available include symbol libraries and some methods of storing non-graphical attributes ('properties') so that parts lists may be produced automatically.

2.1.1.2 2D Applications Modules

These contain extra facilities (which may be used in conjunction with the general 2D module) that make the CAD system more efficient when used in one particular field. Typical examples of modules include:

a) Architecture - additional facilities allowing walls, windows, doors etc. to be created with the minimum of user input.

b) Reinforced Concrete - facilities to design R.C. structures and detail drawings according to particular national standards.

c) Mapping - a mapping database allowing multi-user access and additional facilities to allow use of mapping grid reference systems.

2.1.1.3 General 3D CAD Modules

Four different types of 3D system may be available, each with its advantages and disadvantages, which make each type applicable to different applications.

a) Wire Frame - Lines and arcs ('wires') are defined in 3D space.

Uses : Visualisation, Technical Illustrations, Schematics applications.

Advantages : Fastest of 3D systems, Natural extension to 2D CAD. Can run on all types of computers/terminals.

Disadvantages: Display of complex models can look confusing to user.

b) Simple Surfaces - Planar, Cylindrical and Conical Surfaces.

Uses : 'True' visualisation with hidden lines/surfaces removed. Realistic Images with use of colour and shading.

Advantages : Reasonably fast. Wire Frames and Surfaces can easily be displayed simultaneously.

Disadvantages: Limited surface types. Colour graphics terminals are normally necessary. When modelling solids, 'correctness' of the model cannot be guaranteed.

c) Solid Model - Logical operations (e.g. Union, intersection) on solid primitives (block, sphere, torus, cylinder, cone) used to build up complex models.

Uses : Visualisation.
Realistic Image Generation
Parametrised Objects
Exploded assembly diagrams.

Advantages : Model is guaranteed correct and consistent, enabling sectioning, volume and moment of inertia calculations.

Disadvantages: Can be very slow. Requires more powerful computer. Cannot handle complex surfaces. User interface is more complex.

d) Sculptured Surface Model - Mathematically complex surfaces defined using sections and tangents between sections.

Uses : Mathematically complex surfaces, e.g., Ships hulls, aeroplane wings, plastic mouldings.

Advantages : Interface to NC systems.
Reasonably fast.
Can be easily linked to 2D/3D modules.

Disadvantages: Not efficient for Modelling of regular surfaces/solids.

2.1.1.4 Finite Element Analysis

A model of the structure is defined using 2D or 3D Finite Elements and the structure may then be analysed to produce such outputs as deflection, natural frequencies, temperature distributions, stress distributions. A powerful 32 bit minicomputer is normally required to support Finite Element Modules.

2.1.1.5 N.C. Programming

Modules allowing graphical programming and verification of machine tool paths and actions, for many different types of operation (e.g. milling, lathing, drilling, nibbling, punching, spark erosion). Profiles may be created using 2D or 3D CAD modules, and the machine tool paths viewed in 2D or 3D. NC machines may be driven directly by the computer (CNC) or in an offline mode using, for example, a paper tape containing the required sequence of commands.

2.1.1.6 I.G.E.S. Interface

This module allows the graphical data to be converted from the internal data format used by a CAD system to a neutral ASCII file format as defined by the IGES standard. IGES files so produced may be transferred to other computers and processed by another CAD system, to create the graphical data in the format required by that system.

2.1.2 International Standards for CAD

Standards are useful in many industries, and the CAD industry is no exception. In recent years a few standards have been emerging for CAD, and two were discussed in detail - IGES and GKS. For each standard, the reasons for the standard were first discussed and then examples shown of the practical application of such standards.

2.1.2.1 IGES

Many different CAD systems exist, each with its own unique data structures in which the graphical and non-graphical information is stored. With the widespread use of many different CAD systems, it became apparent that some means of efficiently transferring both the graphical and non-graphical data was necessary. IGES (Initial Graphics Exchange Specification) defines a standard which may be used to perform such a transfer. The standard gives details of a format which may be used to represent different graphical entities (e.g. line, arc, text) in a text file which may then be transferred (e.g. using magnetic tape) to other computers and other CAD systems. The use of IGES to transfer drawing data between two systems (System A and System B) then becomes a three step process:

1. Using System A, run a 'pre-processor' to convert a drawing from the format used by System A, into an IGES file.

2. Transfer IGES file to a computer on which System B is running.

3. Using System B, run a 'post-processor' to convert the IGES file into a drawing file in the correct format for System B.

In actual use, the process may not run quite so smoothly as outlined above. Information may be lost in the transfer process, due to ambiguities in the IGES specification document. The mechanisms for transferring non-graphical data are not of practical use, and symbol libraries cannot be transferred. Another problem concerns the size of the IGES file. which may typically be 10 times larger than the original drawing (and up to 50 times larger). Recently, a new revision of IGES has been announced, Version 3.0, which attempts to sort out these and other problems, and over the next year or so it is likely that most CAD suppliers will announce support of Version 3.0 in place of the current version, 2.0.

Recommendations were also made concerning the following:

1. Method of development of an IGES processor.

2. Use of IGES to transfer drawings between CAD systems.

In summary, IGES offers a viable method of transferring data between CAD systems, provided some care is taken initially when deciding which types of graphical information are to be transferred.

2.1.2.2 GKS

Traditionally, each program which made use of graphics terminals would include a number of subroutines to interface to the particular terminals required. Such subroutines would need to be modified each time a new terminal interface was required, and the form of the subroutines might be completely different for separate programs. This approach was not only wasteful of time and effort, since new graphics subroutines were required for each new program, but also tended to produce complex code where many different terminals needed to be supported by one program. A more sensible approach was to create a self contained library of graphics routines which could be used by any application program to drive many different terminal types, so that the applications program did not need to communicate directly with the individual terminal types. Third party software houses began to offer such graphics libraries commercially (e.g. TEMPLATE, AD-2000), and the logical conclusion of this process was the establishment of International standards for such libraries. One of these standards (and the most widely used), is GKS - the Graphical Kernal System.

Various aspects of GKS were examined, including the available primitives, attributes and segment capabilities. The deficiencies in the standards were also discussed. Notwithstanding these deficiencies, use of such standards is to be encouraged because of the advantages of this approach:

1. Terminal independent software packages calling GKS routines.

2. Availability of GKS libraries from third parties, eliminating the need for production of such libraries in-house.

3. Application software will not become obsolete or require modifications when new generations of terminals are available.

2.1.3 Software Portability

The advantages of terminal independence described in the section 2.1.2.2. apply equally to computer independence. If software can be written in such a way that applications need not call computer dependent subroutines directly, then the software may be ported onto different computers with relative ease, and will run in an identical manner on each computer. Here, however, there are no well defined international standards, and so libraries of computer independent routines need to be designed by first identifying those facilities which are machine dependent and then deciding what subset of those facilities might be required by the target applications.

Machine dependent facilities include:

- Communication to/from terminals, peripherals, processes, files.
- Data Formats
- Systems Information
- Process coordination and Manipulation.

When a machine independent library has been created, it is necessary to ensure that all other coding produced is not in any way machine dependent by calling the library routines where necessary. In order to guarantee machine independence, the following points need to be noted.

1. Programming standards must be defined and adhered to.
2. Documentation describing machine independent libraries and programming standards should be widely and freely available to staff.
3. Education as to the reasons for, and importance of, machine independent coding should be given to all new staff, and on a continuing basis, as necessary, for existing staff.

2.1.4 Graphics System/Database Design

Many different aspects of the design of CAD systems and their databases were considered in detail. Examples were taken from existing CAD systems including DOGS and DOGS3D, to show the sort of decisions which need to be made when a graphics database is designed. Several important points were noted, which need to be considered during the design phase, including:

- separation of graphical and non-graphical information
- separation of graphical data into views or layers
- design for speed of access to graphical data during certain database operations (e.g. tolerancing, windowing)
- design for flexibility, expandibility

- parametric (macro programming) capabilities
- identification and specification of separate modules
- design of the user interface
- the use of a relational database for storage and manipulation of non-graphical attributes.

The importance of software development procedures for all phases of the development process was also stressed.

2.1.5 Management of Large Software Projects

A 'large software project' was defined as

a) One involving 4 or more staff or

b) One involving 1 or more years development effort or

c) One consisting of many intercommunicating modules

or any combination of the above. Large projects pose many management problems, not all of which may occur in smaller projects where a more informal approach might be used. A great deal of time was spent describing and discussing many aspects of Software Project Management, as detailed in the following sections.

2.1.5.1 Organisation and Choice of Staff

Examples were given of typical staff structures, with programmers organised into sections to deal with particular aspects of a project, and sections combined to form groups. The different skills required by Group Leaders, Section Leaders and Programmers was discussed, together with the role of specialists working alongside the Group structure. The importance of recruiting staff with the required skills was emphasised.

2.1.5.2 Estimating

Estimating is one of the biggest problems that programmers and managers face. Programmers are notorious for their inability to estimate timescales for developments accurately, leading to the well known 90/90 rule which states that:

'90% of the development will take 90% of the time, the remaining 10% of the development will take another 90%!'

There are two types of estimating which need to be done:

a) Estimating the amount of work to be done (requirements estimating)

b) Estimating the resources available to do the work (constraints estimating)

It is important that (b) does not influence (a), i.e. the amount of resources actually available should not determine the estimate of resources required. When this occurs, the estimate of requirements is invariable over optimistic. The resources available covers both the hardware - computers, terminals etc, and the programming staff. It is obviously important to have enough of both types of resources - not enough hardware and the staff will be kept idle due to lack of terminals or poor response from the computer.

The lecture concentrated mainly on aspects of requirements estimating. The number of lines of code (NLOC) required for a development should be estimated. Given a figure for this and an estimate of the average number of lines of fully debugged and tested code that one programmer produces in one day enables us to estimate the total time a development will take. Sources indicate that the industry average is between 5 and 10 NLOC per programmer per day, a figure which often appears to be surprisingly low when first encountered, but which has nevertheless been validated many times.

In order to estimate the NLOC, each part of the development needs to be considered in detail. In fact, a significant amount of the total time taken for a development should be taken up in specification of the development to greater and greater detail before a line of code is written. It should also be noted that testing will also take up a large proportion of time, and figures taken from successfully completed large software projects suggest that each of the phases of specification, coding and testing should take approximately one-third of the total project time.

As the project is defined, milestones should also be defined, as well as 'inch pebbles' (much close together than milestones!). These will allow us to monitor the project closely, and adjust our estimates or our resources as necessary as each milestone passes.

One other important point to remember is that a programmer will not spend 52 weeks in every year programming. Significant amounts of a programmer's time will be spent on other activities, including holiday, sickness, training, general administration, supervision of others. The total number of weeks remaining for programming may well be 40 or less. By considering what values are appropriate for an organisation, it is possible to calculate a 'development factor' for programmers which, when multiplied by the estimate of the number of programming days required, gives an elapsed time, i.e. the time the development will actually take.

2.1.5.3 The Software Development Environment

Software tools are essential for large software projects, in order to make most efficient use of what is normally the scarcest resource - the programmers themselves. The following tools were discussed in detail.

- Full Screen Editors

- Debugging Tools

- Source Management Systems

- Integrated Programming Support Environments.

All these tools will use up additional processing power, to greater or lesser degrees, but overall the benefits remain. The programmers can spend more of their time programming instead of housekeeping. The end result is better code, which is developed more quickly.

2.1.5.4 Use of Walkthroughs

Walkthroughs, or Reviews, are a very important aspect of project management. A Walkthrough may be defined as "A meeting of a group of people to review a product with the purpose of finding deficiencies in it."

Some points to bear in mind are listed below:

1. It is the product which is on trial, not the producer.

2. Use a checklist of likely problems, based on past experience.

3. At least 3 or no more than 7 people should attend a review.

4. At least one person present at the review should be from outside the team responsible for the product under review.

5. A Walkthrough should not last for more than half a day.

6. Walkthroughs are an excellent place for new team members to learn about the product and software development standards used.

Walkthroughs may be used at many different phases of a project, including specification, coding and testing. Use of walkthroughs will always result in a better quality product, and are strongly recommended.

2.1.5.5 Testing

Testing is often regarded by programmers as the aspect of software development which is enjoyed least, and there is often a temptation to get testing completed as quickly as possible. This philosophy leads to release of software which has been inadequately tested. Testing should, in fact, make up as much as 30% of the total effort put into a development. The testing effort falls into two phases:

1. Module testing -

   Each module (subroutine, option) is tested independently, to ensure that it works in the way it is expected to work. At this stage, every possible statement within the module should be executed at least once, and 'real life' data should be used whenever possible.

2. Integration Testing -

   This phase of testing checks whether the modules work as expected when integrated into the complete system, and also checks whether the rest of the system still works in the same manner as it previously did. At this stage, tests which were used with previous levels of the software may be rerun.

Whenever possible, automated testing methods should be used, to allow as much testing as is feasible to be undertaken.

It should be remembered that no amount of testing will compensate for a bad software design.

2.1.5.6 Quality Assurance

The best way to ensure that the final product is of a high quality is by the formation of an independent Quality Assurance Team and the production of a Quality Assurance Plan.

The Q.A. Plan should contain a formal definition of all phases of a software project, from the initial proposal through to the ultimate release of the software and on to the maintenance phase. Figure 1 shows an example of such a Q.A. plan. The Q.A. Team should be responsible for the production of the Q.A. Plan, monitor its use and test the products created using the plan. The team should also make observations and recommendations to further improve the quality of software products.

SOFTWARE DEVELOPMENT PROCEDURE

1.	INITIAL PROPOSAL	- Brief outline of proposed development.
2.	ESTIMATE	- Statement of key features. Rough estimate of cost for comparison purposes. Note of features of competitors products.
3.	PROJECT LAUNCH	- Decide who is to be involved in the various stages.
4.	PRELIMINARY SPECIFICATION	- To include draft User Manual. Normally one issue only.
5.	DRAFT SPEC./WORKING SPEC.	- Second issue to be frozen as working specification. High level program design. User Manual from working specification to word processor. Overall Test Specification.
6.	DETAILED SPECIFICATION	- Low level program design. Local Test Specification.
7.	CODING	- Includes appropriate documentation
8.	LOCAL TESTING	- To Local Test Specification
9.	DEVELOPMENT INTEGRATION	- Bring all developments into new level.
10.	OVERALL DEVELOPMENT TESTING	- To Overall Test Spec. Q.A. bug "Snapshot" } ALPHA-SITE TESTING
11.	SYSTEM INTEGRATION TESTING	- Run tests from previous levels
12.	DEVELOPMENT RELEASE	- Ensure all documentation complete. Release to Q.A.
13.	Q.A. TESTING	- Q.A. Section test the development. "Release Report" issue.
14.	BETA-SITE TESTING	- May be in part concurrent with Q.A. testing.
15.	FULL RELEASE	- Release to Customer
16.	PROJECT REVIEW	- Post-mortem on project
17.	MAINTENANCE	

Figure 1 - Example Summary of Q.A. Plan

2.1.6 Applications Generators and CAD Macro programming Languages

Macro Programming Languages were discussed by looking at an example of such a language in some detail - the DOGS Parametric Symbol Language. This language allows the CAD users to write their own 'programs' consisting of commands to select DOGS options, read cursor locations, text etc., and also process information using a language similar to Basic. The parametric language has been found to be very popular with users, who can use the facilities to tailor the standard CAD package for their particular fields. The language, therefore, is also an applications generator. The popularity and success of this approach indicates that such languages should be an integral part of any CAD package.

2.1.7 Involving the User

The advantages of close liaison with users was discussed. Too many software designers do not take enough notice of user requirements, with the end result being that the software may be awkward (or impossible!) to use in practice. The first users of a new package are especially important to the success or failure of the package, and it is important to build close working relationships with potential customers early during the development of a new project. The role of User Groups and their relationships with software suppliers was also discussed.

2.1.8 Implementing and Tuning CAD Systems

CAD Systems which are to be run on many different computer models and terminals need to be designed with portability in mind. This will allow implementations on new machines and terminals to be produced with the minimum of effort.

The actual process of installing software on a customer's machine also needs to be considered. For packages which have many hundreds of users, the installation procedures need to be as automatic as possible, in order to make efficient use of available resources.

When a program has been installed, it may need to be 'tuned' or optimised, in order to run efficiently. A rule of thumb states that '90% of run time is spent in 10% of the code', and it is these heavily used areas of code which need to be identified and optimised.

There are many different ways in which a program might be optimised. The two most common ways are:

1. Eliminate unnecessary I/O (e.g. disk reading/writing).

2. Changing system parameters (e.g. disk sector size, buffer size).

There are, however, some possible drawbacks of optimisation:

1. The optimised code may be less portable, less maintainable or less flexible.

2. Optimising exploits some observed feature of the running of a program (e.g. most usual pattern of user inputs). If that pattern changes, or is not valid for all users, the system may change from being an optimal one, to one which is worse than ever.

2.1.9 Software Maintenance

A project is not complete when the software is installed at a customer's site. The project is then at the start of its maintenance phase. Industry figures suggest that maintenance may in fact take up as much as 70% of the resources of the total project. Staff need to be allocated to a special Maintenance Team to investigate any problems reported by users and fix them where necessary. One problem is that the people who are most familiar with a particular program are likely to have moved on to new developments and will not be permanently available for maintenance work. One solution to this problem is to have a Maintenance Rota so that all development staff are regularly working as part of the Maintenance Team for 2 or 3 week periods 2 or 3 times a year. This ensures that the Maintenance Team contains staff with the necessary skills to sort out any problems which might arise with a package.

One point to note here is that almost all user reported problems are not software problems. Typically, 90% of problems are due to user error and less than 0.25% are genuine software errors. It is therefore important to weed out as many as possible of the other types of errors before passing the remaining ones to the Maintenance Team for further investigation.

2.1.10 Future Trends in CAD

By looking at the recent history of computing and CAD, a number of predictions were made concerning possible future trends. It should be noted that these predictions represent the personal views of the author, and do not necessarily represent the views of PAFEC Ltd. The main conclusions are given below.

1. Memory will become cheap and plentiful (16 Mbit chips by 1990).

2. Faster, 32 bit computers will become widely available at less than $1,000.

3. Future generations of computers will use parallel processors for greater speed.

4. High resolution, colour, engineering workstations with 3D capabilities will become widespread and cheap.

5. Unix (and C) will be adopted worldwide.

6. The emphasis for CAD systems will be on highly user friendly systems running on high resolution, colour, personal engineering workstations.

2.2 Other Training

During the mission, two one-day seminars were organised, one in Varna and one in Sofia. The seminars were presented to invited delegates from institutes and other UNDP projects. A total of forty-five people attended these seminars.

The seminars were entitled 'Introduction to CAD'. A series of lectures was given under the following headings:

- Advantages of CAD
- Introduction to 2D CAD
- Introduction to 3D CAD
- The CAD/CAM/CAE Link
- International Standards in CAD
- Future Trends in CAD.

A lively question and answer session followed the lectures, to end the day.

3. Observations during the Visit

The CAD Laboratory has expanded considerably since Dr. Shaw's visit in 1984. The staff have a wide variety of backgrounds and computing skills, which is certainly necessary for the development and enhancement of CAD systems. What is needed now is a longer term plan for the work which is to be carried out at the Laboratory, so that all staff have a clear idea of their roles within the development plan.

The lack of availability of 32 bit computers and graphics terminals due to the U.S. embargo may be a problem if it continues after any development work has started, since 32 bit computers and good graphics peripherals are a necessity for new CAD development.

The level of interest in CAD shown during the seminars shows that Bulgaria is ready and waiting to use CAD in many different areas of industry. A question remains as to whether all the necessary software can be developed within Bulgaria to meet the requirements of all the prospective users.

4. Conclusions/Recommendations

Bulgaria needs CAD, especially if it wishes to participate competitively in international industrial markets. A decision needs to be made as to whether the CAD software required by industry can be wholly or partially developed within Bulgaria. It is the author's opinion that the software cannot all be developed within Bulgaria in the necessary timescales. A survey of the needs of prospective users would show whether a CAD system could be produced to cover some common requirements, but this system would still need strong links to software supplied from outside Bulgaria for specialist needs (e.g. surface modelling).

It is the author's opinion that with the widespread availability of 32 bit micro-processors, 32 bit machines suitable for CAD systems will be available in Bulgaria within one to two years. Until this happens, however, development and use of CAD software will be severely restricted.

The CAD Laboratory needs to consider the needs of Bulgarian users, and draw up a long term plan to meet those needs, with a mixture of imported and in-house software, within the timescales required by the users.

References

1. Report by Expert 11-02 Senior CAD Advisor Mr. Keith Shaw DP/BUL/81/009.

Document History

Date	Issue	Comments	Author
08/03/86	1.0	Written	MDV