Development and maintenance of design-supporting software

Autor(en): **Werner, H.**

Objekttyp: **Article**

Zeitschrift: **IABSE reports of the working commissions = Rapports des commissions de travail AIPC = IVBH Berichte der Arbeitskommissionen**

Band (Jahr): **31 (1978)**

PDF erstellt am: **12.07.2024**

Persistenter Link: https://doi.org/10.5169/seals-24896

IABSE COLLOQUIUM on:
AIPC "INTERFACE BETWEEN COMPUTING AND DESIGN IN STRUCTURAL ENGINEERING"
IVBH August 30, 31 - September 1, 1978 - ISMES - BERGAMO (ITALY)

Development and Maintenance of Design-Supporting Software

Developpement et maintenance de programmes de calcul de structures

Entwicklung und Wartung entwurfsunterstützender Programme

H. WERNER

Professor, Dr. Ing.

Techn. Univ. Munich

Munich, German Federal Republic

Summary

Mature design-supporting software has to pass through four important phases of development:
1. design 2. Coding and test 3. Pilot installation(s) 4. Marketing, maintenance and enhancement. To neglet one phase endangers the success of the next one. From practical experience, the author states the aims of these development phases and offers advice on how to reach them.

Résumé

Les programmes de calcul de structures doivent passer par quatre phases pour être véritablement opérationnels:
1. Concéption 2. Realisation et essais 3. Installation pilote 4. Marketing, maintenance et développement. La négligence dans une phase compromet la suivante. Des remarques et reccomandations, résultant d'expériences pratiques, sont faites pour chacune de ces phases.

Zusammenfassung

Marktreife entwurfsunterstützende Software durchläuft vier wichtige Entwicklungsphasen:
1. Entwurf 2. Erstellung und Tests 3. Pilotinstallationen 4. Vermarktung, Wartung und Weiterentwicklung. Vernachlässigungen in einer Stufe stellen die erfolgreiche Durchführung der nächsten in Frage. Ausgehend von praktischen Erfahrungen werden Anmerkungen und Empfehlungen zu den einzelnen Entwicklungsstufen gegeben.

1. INTRODUCTION

1.1 Definition of Design-Supporting Software

In the process of structural design, the following steps can be defined [1]:

- collection of input information (e.g. preliminary dimensions, loads, construction stages);
- development of a model for analysis (e.g. an FE model);
- analysis
- interpretation and modification of the results according to standards and practical experience;
- calculation of the members;
- construction, i.e. preparation of drawings, details etc.

In each step, information gained by the preceeding steps is used in addition to new data.

Fig. 1 shows a scheme of this process.

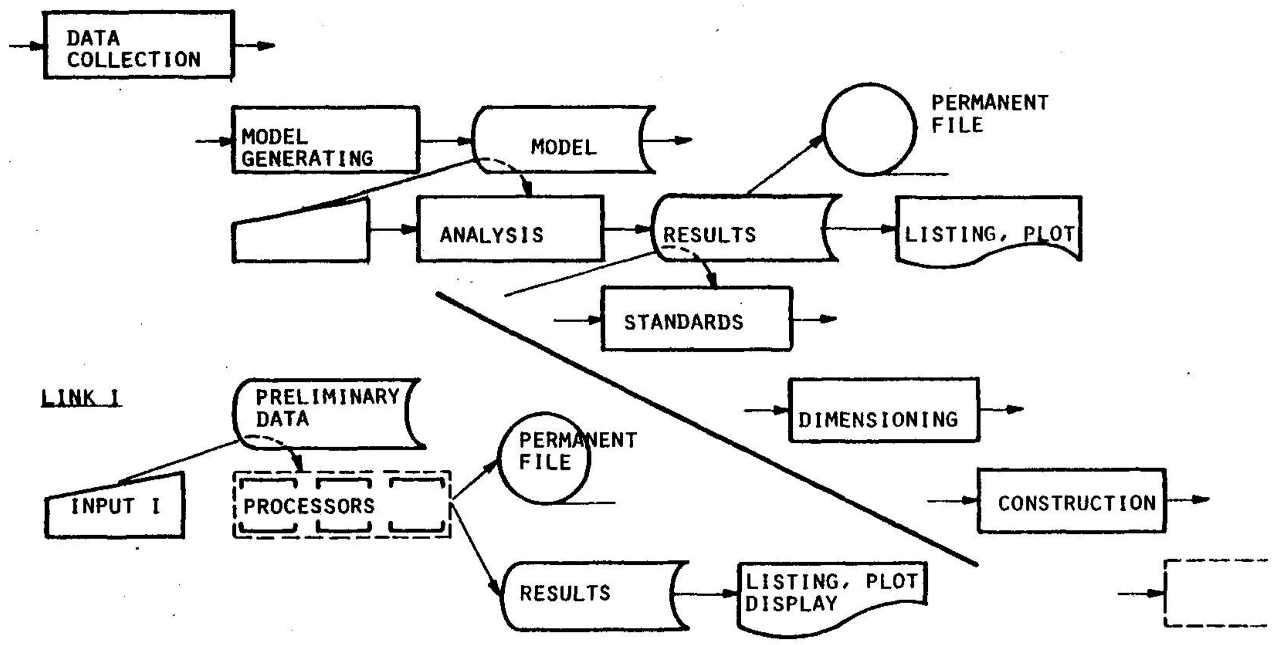

Fig. 1 Scheme of the Design Process

Software meant to automate parts of this chain of processes which are connected with each other to a certain degree could be called "process oriented software".

For every type of construction there is a different process because there are different aspects to the design e.g. of bridges or of soil structures.

Solutions for problems in different processes are often similar (i.e. use of the FEM in structural analysis). Software has been developed to solve special problems (i.e. mechanical or graphical). This type of software may be called "problem oriented software".

Take a matrix which contains horizontally the steps of the respective design processes and vertically the different constructions. CAD-Programs (CAD = Computed Aided Design) are an example of the former group (rows) and FE-Programs [2] are an example of the latter group (columns). The expression "design-oriented software" may be used to comprise both groups.

1.2 Hardware - Software - User Interface

Design-oriented software provides an interface between computers and engineers (fig. 2).

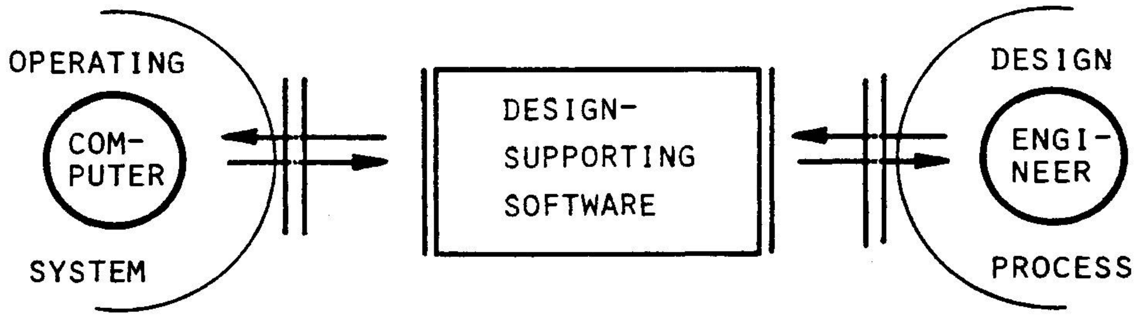

Fig. 2 Interfaces between Computer, Software and Engineer

Software has to be adaptable both

- to the computer and its operating system and
- to the engineering design practice.

These "interfaces" become important in the transition from software "made to measure", for one computer system and one firm, to "standard software", intended for many applications on different makes of computers.

The expression "software", used in this paper, means a complete set of information describing a program, comprising

- short description of the program,
- user manual,
- data processing manual,
- source code.

Hardware exists in many forms; it can be classified approximately into [3]:

- mainframe computers,
- minicomputers,
- desk-top-calculators or microcomputers,
- terminals (without any computing capabilities)

Computers of different sizes are increasingly linked together, i.e.

- terminals are used as peripherals for micro- or minicomputers or connected to commercial timesharing-systems via telephone lines,
- microcomputers serve as intelligent terminals to prepare data for larger computers,
- minicomputers and mainframers work together in computer networks (i.e. minis as front-end-processors).

The smaller (and cheaper) a computer, the greater the number of installations; design-supporting software consequently has to meet the needs of mini- and microcomputers in order to find an adequate market.

Similar to the hardware hierarchy, engineers can be classified according to their knowledge of data processing. The majority of structural engineers are strangers regarding the requirements of hardware and the methods and models of software. They need the help of "interpreters" once their involvement is beyond that of theoretical background.

Much rarer is the engineer who combines knowledge on software applications and structural design, the "finite element engineer". Therefore, software must be aimed at for the "normal" engineer in order to attain wide distribution.

1.3 Software Examples

In the following chapters two software examples will be used for illustration:

1. SET - a chain of programs for geotechnical problems [1]. SET contains components to

   - generate structural and system data (GENSET);
   - analyse FE-structures with non-linear material behaviour (NONSET);
   - calculate reinforced concrete members, i.e. tunnel linings, tied-back walls (CONSET);
   - analyse seepage and groundwater movement (SISET);
   - print and plot the results (PRINSET, PLOTSET);

   All components are linked via a common data base (fig. 3).

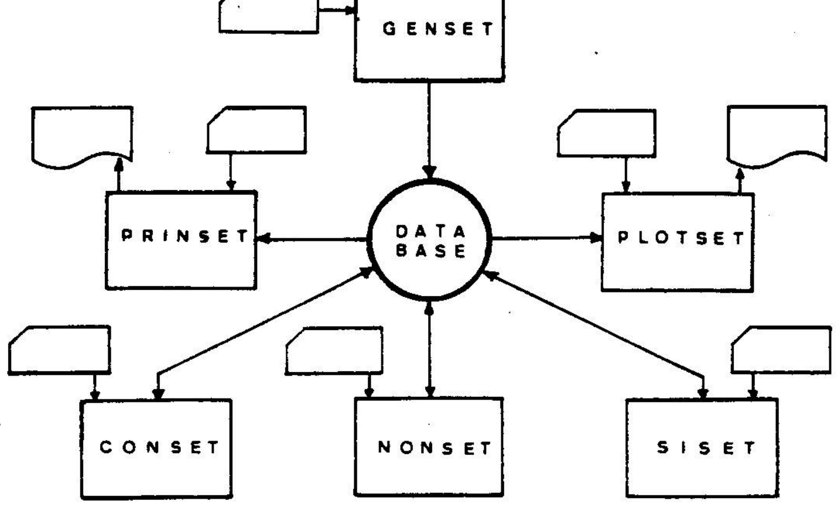

Fig. 3 Components of Program SET

2. HOPP - a program package for the structural engineer.

As a consequence of the wide distribution of desk-top calculators and of their growing performance, engineers require software for their run-of-the-mill problems; HOPP, a package in BASIC, fits this need. Standardization of input (interactive), output and interfaces for all components has led to its wide distribution.

Components exist for the analysis of RC beams, slabs, columns, foundations, plane frames etc. and perform structural analysis, superposition of member loads and dimensioning according to the standards.

2. STEPS OF SOFTWARE DEVELOPMENT

Software development comprises four stages from the problem definition (a description of the desired capabilities, the methods and the "boundary conditions" of a program) to the mature product:

- program design,
- coding and tests,
- pilot installation(s)
- marketing, maintenance and enhancement.

Each stage must be completed before the next one is started; omissions and carelessness in one stage lead to expensive corrections in the next or endanger the whole project.

In [4] a great number of topics to be considered has been summarized.

3. PROGRAM DESIGN

According to the German CAD standards [5], concepts must exist for

- input,
- output,
- solution methods and algorithms,
- program structures and data flow,

before coding can begin. The results of this first stage of software development should be fixed in provisional "User's Manuals" and "Data Processing Manuals".

3.1 Input

Input and output are the interface between design-supporting software and user. A user must not be asked to adapt to new input forms for every new program; a strange form of input can build a barrier against the introduction of new software. An interface between the input component and the rest of a program allows the adaption of the input component to a form the user is familiar with. Some firms do well with traditional input (with fixed columns) - especially, where large amounts of data have to be transcribed by data typists from form sheets to storage media. On the other hand, free format input using a problem oriented language [6] leads to ease of use on terminals. "Guided" input is useful for a dialog with the computer if it is coupled with immediate validity and plausibility checks. This gives guidance to the engineer who does not constantly use the computer but may be superfluous for the experienced and constant user.

3.2 Output

The results of a calculation should be preserved in a storage file; this is advantageous for the following reasons:

- a subsequent program can use these values as input (fig. 1);
- on a permanent file, the results can be kept as long as they are needed;
- subsequent processes, i.e. superposition of load cases, can be started immediately;
- similar to many input forms, several forms of output can be selected (fixed or free formats, degree of condensation, printed or graphical output).

In Munich work is currently being done on a program which lets the user choose the contents and form of the printed output.

On the subject of print output, some obvious facts are mentioned here because they are often overlooked:

- To ensure easy transition between batch mode and dialogue, line numbers should be counted and the number of lines per page should be limited by a variable (which can be set according to the printer);
- every page should automatically be marked by a heading (project identification, date, page number etc.);
- print output should adhere to standard paper sizes (in Germany: DIN A4);
- empty lines waste paper without leading to greater clarity;
- every print output must be self-explanatory without the help of manuals etc.

3.3 Solution Processes and Algorithms

With the application of data processing in design practice, many new solution processes and algorithms have been developed. The practical engineer as program user is often not familiar with these methods; he must, however, check the results and take responsibility for them. Here we have a difficulty which we could meet by the following means:

- Full documentation of the theoretical background of a program, in such a way that the manual becomes a textbook complete from references to the actual implentation;
- introduction of an adviser between software and user. This could be an engineer familiar with the dp methods in a firm, a computer center or an institution concerned with marketing software.

If a new technical model or a solution algorithm is to be introduced, it must be checked for numerical stability, limits of applicability and practical reliability. Such research ought to occur before a design-supporting program is planned. Only tried and proved algorithms should be used for broad (and often uncritical) applications. On the other hand, new techniques are necessary. Their development must be supported if software development support is to be of long-term effect. The result of such research cannot be mature programs but algorithms, which are turned into programs for practical use in the next step (i.e. SAP IV). If the result of such research is a source program, then this source program should be published together with all necessary explanations.

Design-supporting programs refer to standards, recommendations etc. once they leave the area of classical mechanics. Standards, however, usually do not give an unequivocal answer for every detailed problem; interpretations are necessary. Engineers are confronted daily with interpretations; often they decide open questions by experience and engineering judgement.

Design-supporting software has to decide some of the questions in advance, which puts the responsibility on the program author; he has to fill the holes in the standards. In a way he sets a standard himself because possibly hundreds of users may trust his decision implicitly. This is another problem which must be solved.

Some recommendations are:

- design-supporting software development needs the intensive assistance of engineers versed in design practice;
- interpretations of standards must be documented and the standard-committees must be made aware of the problem;
- software engineers should participate in standard-committees in order to adapt standards to dp needs.

3.4 Program Structure and Data Flow

A program consists of routines; each routine should have a clearly defined task and a clearly defined data interface to the rest of the program. CAD standards[5] recommend a size not exceeding 200 statements per FORTRAN routine. It is therefore necessary to analyse the problem, to structure it into sub-problems, until each routine meets these requirements.

At the same time, the data flow between components of the program must be planned.

Interfaces can and should be defined, not only between input and output components and the rest of the program, but between large blocks of routines as well. For instance, the following program and data structure has served well for FE analysis of construction stages (for bridges and tunnels)[1]:

1. "System data" (nodal point coordinates, material parameters, element descriptions, loads and boundary conditions) are input, checked and stored on a permanent file ("original qualities").
2. The FE model representing a certain construction stage (or design alternative) is built up using pre-defined members; data concerning this "current structure" are stored in a working file.
3. The current structure is analysed; results are printed and stored on a working file.
4. If the user so decides, results are stored with the "original qualities" as "acquired qualities".

In this way, the user can model the construction sequence, analyse alternatives or control non-linear processes (i.e. system creep).

A number of programs can be linked to a chain if the external files are clearly defined.

In order to curb the proliferation of programs it seems to be absolutely necessary to design program components to be interchangeable between different program packages (i.e. FE-solutions or graphics packages).

Expensive interfaces become superfluous when data bases are standardized; in any case general recommendations are very much needed. With SET the author offers an initiative in this direction (see chapter 1.3).

4. CODING AND TEST

4.1 Programming Language Considerations

The choice of a programming language for engineering programs rests largely between FORTRAN and BASIC. Up to now, FORTRAN is the language generally recommended for technical programs because a standardized version (ANSI-FORTRAN) exists giving a high degree of portability. For smaller programs, desk-top calculators programmable in BASIC only, offer the widest market. The problem with BASIC is its lack of standardization. Each computer has its own dialect; this in not prohibitive however, because the popularity of desk-top calculators makes it worthwhile to adapt programs to the different dialects. Experience with the program package HOPP has shown that an adaption to each type of computer is necessary but that these alterations account for only a small percentage of the development effort.

4.2 Reduction of Software Errors

Coding means the translation of ideas (collected in the manuals) into statements in a programming language. In practice, however, more time is spent in testing. Hierarchial structures and clearly defined interfaces make it possible to code and test program segments independently, one at a time. By this method, one can be reasonably assured that an error (or an alteration) in one segment has no unforseeable repercussions in another.

Software errors may be defined as:

- errors in the source code,
- errors in the manuals (i.e. description of input),

Clearly, it is efficient to use routines already tried and proved in earlier projects because the effort for testing can be much reduced.

Testing can be made easier if the search for errors is planned in advance. Valuable tools are:

- clear structuring;
- options to print intermediate values ("trace"); it is helpful to print input and output parameters for routines and data transferred to or from files; it must be possible to pinpoint certain areas to reduce the bulk of data;
- redundant validity checks whereever possible; every rountine should "defend" itself against incorrect parameters.

Errors in the manuals are often caused by program alterations not noted in the manual. For this reason disciplined updating is essential.

Errors in the solution algorithms occur if cases are analysed not considered in the design stage. Here the solution process must be enhanced or these cases must be specifically excluded in input description and by input checks.

5. PILOT INSTALLATIONS

Pilot installations following the implementation stage quickly detect:

- installation difficulties,
- errors not found by previous testing,
- practical cases not concidered in the development.

One of the most important sources of information for the program author is the user feed-back. Every recommendation to extend the scope of the program, every misinterpretation of the manuals, every question as to the theoretical background (or to the use of the program or to the meaning of the output) has to be analysed carefully and will usually lead to alternatives in code or manuals. Changes in large FE programs (i.e. ASKA, NASTRAN) are tested for one year by pilot installations in large and experienced firms before being released to the public.

Pilot installations involve a lot of human effort and are therefore expensive; these costs have to be planned into the budget.

6. MARKETING, MAINTENANCE AND ENHANCEMENTS

Only a few large firms have the means to develop design-supporting programs on their own; hardware firms have more or less stopped to develop application software. These days most software must be bought.

The customer is confronted by the following problems:

1. Orientation: Often the information concerning software is insufficient.
2. Choice: Reference lists of users probably provide the best judge of performance and reliability.
3. Installation: This can be particularly difficult with software designed for a special type of computer. Once again, references are helpful.
4. Training of staff: Effort here can be reduced considerably if standards in manuals [5], in input form [6] and in output are generally accepted.
5. Maintenance: Correction of errors needs effort in direct relation to the number of applications. Software has to be adapted constantly to changing standards, techniques or solution processes, if it is not to become obsolete. If software is accepted by the engineers of a firm, then the number of applications grows and also the demands for expansion of the original scope of a program. Who is to maintain and evolve a program? To do it in one's own firm, the user needs dp staff who are willing and able to analyse the program. Otherwise, maintenance should be done by the vendor or the author.
6. Training and advice: New software often contains new algorithms, different interpretations of standards or new theoretical background. Users must familiarize themselves with these novelties and decide whether a new program is applicable to their problems. In this they need help.

These questions demand answers from the vendors:

1. New software products must be published. Mere descriptions are not enough; examples of practical applications awaken interest. Software lists, short information, publications in engineering journals are possibilities. A lack of suitable marketing organizations is obvious.
2. Sample runs of data prepared by the prospective user are often expensive; on the other hand they are a welcome addition to the testing process.
3. Installation effort is reduced considerably if
   - ANSI-FORTRAN is used,
   - the data processing manual contains clear information concerning the installation procedures.
4. User's access to new software can be simplified by:
   - detailed examples in the user manual(first program applications usually take their pattern from examples);
   - adaption of input to a well-known form, i.e. by use of form-sheets familiar to the staff.
5. Software always contains errors; every new installation brings new maintenance problems. Maintenance and enhancement must be guaranteed. This stage of development never stops once a program is on the market; product without maintenance is soon obsolete.
6. The author of software acquires new knowledge in the process. As a natural consequence he has to give advice and publish his new knowledge. On the other hand, councelling by phone can be time consuming, especially if the program is widely used. A requirement becomes necessary for somebody who can answer standard questions and give normal advice to customers while retaining valuable hints for transmission to maintenance staff. This person should ideally be with the marketing organization.

These activities are quite time consuming and expensive. They increase with the success (i.e. distribution) of software. The development including the pilot installation accounts for about 25% of the costs, the rest is spread over the above mentioned points. Approximatively 1 - 3 years time lag occours between the end of phase 2 (coding and test) and the first sales.

7. CONCLUSIONS

As mentioned before, design-supporting software is never finished; it needs constant maintenance and constant adaptions to changing surroundings. If a program is funded in some way, gains from sales etc. should be fed back into maintenance and expansion.

Software engineering is a young disclipine certain to play a key role in future scientific and technical prograss. It should be supported and helped to stand on its own feet. On no account, however, should subsidizing public agencies shackle the free development by unnecessary restrictions.

ACKNOWLEDGEMENT

The author wishes to thank Dipl.-Ing. K. Axhausen, member of the Group on Electronic Computation in Structural Engineering of the TU Munich, for his collaboration on this paper.

REFERENCES

1. WERNER, H., AXHAUSEN, K., KATZ, C.: Programmaufbau und Datenstrukturen in entwurfsunterstützenden Programmketten. Tagungsbericht: Finite Elemente in der Baupraxis, Hannover, 1978
2. DUNDER, V.F.: Finite Element Programs or Software Engineering Library? Proceeding of the 4th Intern. Seminar on: Computational Aspects of the Finite Element Method, Long Beach, 1977
3. NOPPEN, R.: Technische Datenverarbeitung bei der Planung und Fertigung industrieller Erzeugnisse. Informatik-Fachberichte Nr. 11: Methoden der Informatik für Rechnerunterstütztes Entwerfen und Konstruieren, Springer-Verlag, Berlin, 1977
4. BLAND, R.L.: Engineers and Computers-Interaction or Reaction. 6th National ASCE-Conference on Electronic Computation, Atlanta, 1974
5. LANG-LENDORFF, G.: CAD-Guidelines, CAD-Bericht KFK-CAD 6. Gesellschaft für Kernforschung, Karlsruhe, 1976
6. AHN, M., BÖCKELER, K.H., HAAS, W.: Eingabekonventionen für CAD-Programme. CAD-Bericht, KFK-CAD 39. Gesellschaft für Kernforschung, Karlsruhe, 1977