13. PRACTICAL SOFTWARE USABILITY CRITERIA

13.1 Introductory remarks

Having run the programs mentioned in the previous chapters, as listed in Annex 1, the question arises if criteria can be developed for software that is appropriate for the irrigation practitioner. That question of software quality and criteria for quality is the subject of this chapter. The contents are a mixture of a limited number of references, some floating initial ideas about what a good program is from the user point of view, and experiences with the tested programs.

When discussing software criteria, it should be established who decides the quality aspects to be evaluated, even if one tries to avoid personal preferences as much as possible. It is possible to distinguish between various groups of people who can judge software quality, such as the developer, the maintainer, the buyer and the end user. Each of these may have a different perception of what is important in terms of quality. The developer and the maintainer fall in the category of software engineers, which is outside our present scope. We would be more interested in what the end user (which in our case is also often the buyer) would consider to be the necessary qualities of a good computer program.

It is possible to distinguish 15 software quality factors, which can be grouped into two categories of the user’s operational and maintenance needs (Deutsch & Willis, 1988). Leaving the maintenance needs aside, the two major concerns of the user regard functionality and performance. Deutsch & Willis ascribe four software quality factors to each of these two concerns, viz.

<table>
<thead>
<tr>
<th>Functionality:</th>
<th>Integrity</th>
<th>How secure is it?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reliability</td>
<td>How often will it fail?</td>
</tr>
<tr>
<td></td>
<td>Survivability</td>
<td>Can it survive during failure?</td>
</tr>
<tr>
<td></td>
<td>Usability</td>
<td>How easy is it to use?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance:</th>
<th>Efficiency</th>
<th>What resources are needed?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Correctness</td>
<td>Does it comply with requirements?</td>
</tr>
<tr>
<td></td>
<td>Safety</td>
<td>Does it prevent hazards?</td>
</tr>
<tr>
<td></td>
<td>Interoperability</td>
<td>Does it interface easily?</td>
</tr>
</tbody>
</table>

In this concept, stemming mainly from the software engineering discipline, we would be mostly interested in the Usability factor, which deals with the initial effort to learn, and the recurring effort to use the software for its intended purpose. Usability can e.g. be enhanced or degraded by the naturalness of the user interface, the readability of the documentation, the logic of the program structure and the number of keystrokes required for a given command.
When discussing usability, there is an obvious link with the target group of the software. We have called this group the irrigation practitioner in Chapter 1. This means that a solid training in irrigation is present, and that only a rudimentary knowledge of microcomputers is assumed, referring to an understanding of the limitations of computer use and a basic notion of equipment and application packages. Above all, however, we assume an interest in the application of microcomputers in areas like (1) analyzing research data, (2) writing scientific research or monitoring and evaluation reports, (3) solving design and operational irrigation problems, and (4) providing education and training. We, moreover, assume that we talk about professionals who do not have access to the latest computer technology and rely on hardware and software that is readily available and of proven performance. This requirement would probably be standard in many smaller institutions, especially in developing countries.

The question of irrigation software criteria was also addressed by Rogers et al. (1991), who were to evaluate unsteady flow models for irrigation canals, so as to be able to provide guidance to potential users on the use and limitations of the same. Program capabilities, application and usefulness were investigated. They distinguished three groups of criteria, viz. technical merit, modelling capabilities, and user considerations. The technical merits concerned e.g. computational accuracy and numerical solution criteria, which do not concern us now. Also, modelling capability aspects like system configuration, turnouts, etc. are outside our scope. However, the user considerations mentioned by Rogers et al. are interesting for our discussion. They state that a computer model is valuable only if the user can apply the model effectively, for which the following aspects are important:

- **user interface:** the interaction between user and computer influences the problem-solving efficiency. Effective data management, consistent terminology, flexibility through interactive execution mode, numerical and graphical output are mentioned as important concerns;
- **documentation and support:** written documentation, a user’s manual, example input files and output, a case study, interactive help functions and error diagnostics, source code availability, technical support by the developer are listed as contributing to useful programs;
- **direct and indirect costs:** the purchase price of the software and required hardware and additional software can be considered as direct costs; often more important, however, are (indirect) labour costs for learning, for data entry and operation, and for interpretation of results. The latter can be minimised by appropriate documentation and user interface.

In this context of usability and user considerations, we concentrate on four aspects, which are discussed below in some more detail, i.e.:

- hardware requirements;
- user-friendliness;
- the manual;
- availability.
It is realized that criteria are not to be considered in isolation, and are not only
determined by the user group but also by time. The latter means that many remarks
made below may be subject to change within a number of years, if the fast
developments in microcomputer technology and use we have seen up to now are
anything to go by. With such reservations, let us look at the four above aspects in more
detail.

13.2 Hardware requirements

Irrigation programs are written for a particular hardware configuration, which not only
concerns the computer itself, but also the peripheral devices like monitor, disk drives,
printer, etc. Let us consider which requirements could currently be considered normal
and which are excessive for our previously defined target group.

* The microcomputer:
The program should be suitable for IBM microcomputers or compatibles; either XT or
AT (286, 386 or 486) machines. The currently common maximum requirement may
specify a 286 machine or a 386SX, although the 486 is gaining ground rapidly. Clock
speeds are not normally indicated as hardware requirement; 16, 20 or 25 MHz should
be sufficient.

* The disk operating system:
The standard disk operating system that is required should be Microsoft DOS (MS-
DOS) and should not have to exceed version 3.3 (or the equivalent DR DOS or PC
DOS). Requiring version 4.01 is still considered asking too much, although version 5.0
(eespecially useful for WINDOWS applications) has already been in use for some time
now.

* The internal memory:
The maximum required internal (random access) memory (RAM) is 640 kB, which was
the standard a few years ago. Asking for 2 MB or more is asking too much of the
common user we have in mind. This virtually excludes the use of WINDOWS.

* The hard disk:
Specifying that a hard disk with 20-40 MB capacity should be present (with at least 1
MB free) is not a particularly excessive requirement, but specifying that 60 or 100 MB
is required is not yet realistic. Hard disks are also characterized by access time, but
specific requirements do not normally occur.

* The co-processor:
For our standard user, the need for a math co-processor (8087 for the XT, 80287 or
higher for an AT) would almost be asking too much. One might say that it comes in
handy (and speeds up computations), but the program should also run within a
reasonable time without it.
The monitor:
Even if a program was developed for use on a colour monitor, it should also be suitable for monochrome displays (either through colour filters, providing suitable device drivers, selecting proper colour settings, etc.). The maximum screen resolution that could currently be requested is VGA (640x480), which will also handle programs based on the earlier Colour Graphics Adapter and Enhanced Graphics Adapter monitors. Requiring Super VGA or higher resolution is still asking too much. The program itself should preferably identify which type of monitor is available and set its switches accordingly. If not automatic, a SETUP for the proper screen type selection should be available. A mouse is not normally an explicit requirement.

* The printer:
If a printer is required, a standard 9-pin matrix printer (like an Epson LX-800, STAR LC-20 and the like) should be sufficient, although 24-pin printers are rapidly becoming more common. Specific plotters can be nice for printing graphics, but should not be required as a standard outfit. The fact that a printer is required to run a program is considered a slight anomaly. In principle, one should be able to do virtually everything on screen, and input data and results (tables, graphics) should be stored in files. One can then decide whether or not to print from these files later. It means that such input and output files should contain meaningful data, with recognizable headings, etc. They should be retrievable by most universal packages like LOTUS, Harvard Graphics, etc. for further processing.

* The floppy disk drive(s):
The presence of a second floppy disk drive (3.5" or 5.25") should not really be necessary (one sometimes sees: A: for the program disk and B: for the data disk). The 3.5" drive is rapidly taking over from the 5.25" drive. Programs should preferably be stored on double density diskettes, to avoid handling problems which may occur with high density ones.

* Other devices:
Digitizing tablets, plotters and other devices may be occasionally required by some programs based on drawing programs like AutoCAD. This is not considered a normal requirement. CD-ROM players, digitizers and multi-media kits are still for the future.

13.3 User-friendliness

This is a generally used but loose term, although it has a more specific contents in software engineering circles. As software users, we distinguish the following aspects of user-friendliness of irrigation programs:

* "Stand alone":
The program should stand on its own: a new user should not need the manual to find his way around in the program under normal conditions. In practical terms, this often
means a menu-driven program, with logical and clear options/choices at the various levels. These days (following the increased use of a mouse) this often means the use of pull-down menu's. The structure and the logic of the menu's should be such that the standard user, with knowledge of the subject, almost automatically finds his way. Help screens provide explanation in different situations (where the manual was needed previously), preferably context-sensitive.

* Screen lay-out:
The importance of the monitor and what appears on it as the communication interface with the user can hardly be over-emphasized. Programs should not have crowded screens with excessive information, nor should they be devoid of any guidance (especially for new users). In most programs there are more levels, and especially a new user wants to know at which level he is, which file he is using, what his possibilities are (including a Quit and Exit or an Escape) and what the consequence of a certain choice would be. Standard but salient catch words, uniformity in lettering, colour and location of certain information (status bar, instruction bar, etc.) should provide ease of operational security and confidence.

* Standard basic actions:
There should be well-considered system in the basic movements on the screen, such as moving the cursor with the four arrows, contracting highlighted choices with <Enter>, leaving a menu with <Esc> (or possibly with F10), the presence of Help under F1, etc. Most of these standard movements are borrowed from the more widespread general application packages like WordPerfect, Lotus 1-2-3 and dBASE. Also the presence of defaults and range limits (avoiding errands into the unknown) are helping the unskilled user to easily find what he wants, instead of discovering this by time-consuming trial and error.

* Fool proof:
The program should have thorough protection against hitting the wrong key accidentally and ending up with a messed up program, loss of data and a computer that is "hanging". As much as is practically possible, the program should be fool proof. A carefully selected set of error messages may also be helpful in this respect, although it should not be overdone.

* Interactive data input:
Data input, an important aspect in the numerical programs in scientific applications, should be structured in an inter-active way, often requiring a special user interface. This requirement avoids that users must count columns and enter blanks and data in a separately specified format, as was the case in many early Fortran programs, which were converted from punched cards input. The program should offer highlighted and commented data fields on screen and should not bother the user with where exactly it stores the entries. Data entry is the main contribution from the user, but it should be made easy, self-explanatory and quick by the program.
* File handling:
   In the same way as the program should facilitate data input, it should also make file handling easy. There should be no need to leave the program (and e.g. go to MS-DOS commands) to update, save, delete, rename or copy input or output files which are needed during program execution or during initial analysis of output. A good program foresees the most common requirements and has them "in house"; it also means that not every odd wish is catered for as this could easily clog the program and thus lead to confusion.

* Graphics:
   The use of graphics for the evaluation of results could also be considered part of the user-friendliness. It is often faster and easier to compare the outcome of two different runs graphically than to do this comparison by looking at data tables. Even though tabulated data would be required for further processing, on-screen graphical viewing adds to the ease and the speed of evaluating what you (and the program) have done. Good graphics, lettering and other techniques can render an enormously important communication dimension to a program.

13.4 The manual

There should always be a manual accompanying a program, even though we said above that it should not be necessary for normal operation of the program. But it is required to document the objectives, target groups, relevant current developments, the methodology and the process of program development, the background theory, the use of approximations and constants. It should also explain the use of the program step-by-step and point out any less common uses. At least one worked example should be included, the data of which should already be available on the distribution disk. Its usefulness is mainly as a reference guide for more interested user groups and evaluators. We are intentionally ignoring the fact that for the author a separate system engineer's manual (and other documentation) may be needed for program maintenance and further development: the common user should not be bothered with these details. Without explaining each and every basic computer handling detail, a good manual should contain at least the following, although not necessarily in this order.

* Introduction:
   An introduction, in which the following topics are covered: a brief history of the program, the program language, the aims of the program, the intended target group, what the program can do and what it cannot do. The minimum required and the recommended hardware configuration (PC, RAM, FDD, HDD, Monitor, Printer) and disk operating system (and minimum version number). If necessary, a few general programming remarks may be made. The files present on the distribution disk should be listed and a contact address for further information and possible comments should also be given. A short description of the structure of the manual should conclude the introduction.
* Background theory:
A concise section in which the subject matter of the program is reiterated. This theoretical background is mainly meant to instill confidence in the reader that theoretical concepts have been followed which are valid and up-to-date. Of course, references to more detailed theory can and should be given, but a first evaluation should be possible without having to go to such references. This section would include a number of basic formulas used in the program, a consideration of boundary conditions and ranges of values for which the program solutions are valid. The inclusion of some tables or ranges of relevant input values is often useful. The choice of any numerical solutions should be explained and any approximations or the use of controversial constants defended. Such a chapter is not only a matter of scientific accountability, but may also serve educational purposes and assist others to further explore the chosen roads.

* Program structure:
An appropriate summary and explanation of the structure of the program, with its various menu’s, levels and options. There should be a short explanation of each feature on the screen in a logical sequence. It is often useful to separate this explanation from the actual use of the program. It can also serve to give reasons why certain choices were made.

* Running the program:
Instructions for the use of the program. One could probably start with installation procedures and then mention general screen movement and other conventions: cursor movement, <Enter> to accept, <Esc> to go up one level, F1 for Help, etc. Then an explanation would follow on how to start, which screens are showing up, main program choices, menu options, data input, error messages, etc. This is the documented equivalent of actually starting up the program and running it by following the screen instructions.

* Example case:
At least one sample problem or case study, which illustrates the normal use of the program. This not only concerns the use of menu’s, choosing options and reading Help, but the case study should also be designed in such a way that most features are shown without confusing the (first) user. Although this could be a useful exercise, data input should not be necessary (if it takes more than a minute or so) and data files should be available on the distribution disk. Including a simple first example and a slightly more complex second one is normally a good strategy.

13.5 Availability

Although this may seem a rather superfluous point, it still is important for a good program that it is available to the intended target group. This means that it should be adequately advertised, it should be sent quickly when ordered, the price should not be prohibitive and there should be a contact person at a fixed address.
Although other categories are possible, one could distinguish between:
- public domain software;
- software developed by public educational & research institutes;
- packages developed by consultants for specific clients;
- packages from publishers and other commercial institutions.

Development costs of all but the simplest computer programs are high and even supplying public domain software costs money (for copying, handling and postage; even bulletin board programs often suggest a contribution). Educational and research institutes mainly operate on public funds and they normally do not charge development costs. The current move towards a withdrawing public sector and the need for government bodies to generate income lead to increasing costs for software packages. Apart from the financial aspect, there is also the psychological barrier in giving away a lot of time and effort to someone else, who could benefit from it in an unethical way. Publishers and consultants charge high prices for their programs in a bid to recover the development costs: they work on a purely commercial basis.

Asking a price (be it high or low) brings about questions of protection of the software package against illegal copying. This could be done by including a copy lock, by allowing a very limited number of copies, by including an expiry date, etc. Institutions (libraries, government institutes, industries) have to respect the copyright, but checking unofficial use is not an easy task, especially not in developing countries. As such, a high price is already prohibitive for restricting normal circulation, but to stop illegal copying requires other protection measures.

For our earlier defined target group, availability often simply means: price. Generally speaking, exorbitant prices cannot be afforded, except under special project funds. A price of USD 5000 for a program is then very high, whereas up to USD 100 would often be affordable. Larger government institutions, and firms working in the private sector, can more easily find money for computerization, and a software budget accompanying the purchase of hardware may more easily accommodate relatively high prices for a program. In such cases a few thousand US dollars for a program may seem completely reasonable. For individuals however, price often remains a major obstacle towards availability.

References


CHAPTER 14

14. SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

14.1 Summary

In this publication we set out to make an inventory of readily available microcomputer programs for the irrigation discipline. After a few introductory chapters, over forty readily available programs were discussed according to a convenient classification into nine sub-areas. In addition to test-run programs, a limited number of literature references to programs were included. Let us first summarize the findings per sub-area and then proceed to list some more general points.

In the Games chapter we noted that the non-computer-based games discussed either have a strong manager focus (JUBA), a project planning bias (RIVER WADU), or an unspecified farmer orientation (SIMBOL). Only the Irrigation Management Game (IMG) has a slightly stronger engineering component. The River Wadu and the IMG may be useful for irrigation engineering courses, especially for team building and related skills. Of the four discussed computer-based games, one game partly aborted (WYEGAME). A second has very nice graphics, but lacks a more general training purpose (SUKKUR). A third (IRRIGAME) does not make the irrigation decisions very clear and is geared towards American conditions, while a fourth (REHAB) also stresses the group experience rather than the irrigation knowledge. It seems useful to follow up on the Mott MacDonald programs NILE and MAHAKALI with their superb graphics.

The Water requirements and scheduling chapter showed that there is a variety of programs dealing with computations of crop water requirements, mainly based on a reference evapotranspiration, although some other relations also occur in American programs. These programs are either single purpose (ETREF, CRIWAR, CRWTABLE), or embedded in scheduling programs (CROPWAT, IRSIS). Such programs also form the basis for various other irrigation scheduling programs which are in use in various countries, either commercially or as an extension service. FAO's CROPWAT has the advantage of a wide dissemination.

The Field irrigation programs have benefited greatly from work by Strelkoff and Katopodes on zero-inertia models. An earlier analytical border flow program (BRDRFLW) got a sequel (SRFR), which is more versatile, but lacks user-friendliness. A useful and user-friendly tool for design and analysis exists (BASCAD), while similar programs did not work properly on our test machine (BASIN, BICAD). Volume-balance approaches are also used, especially in programs from Utah State University (SURFACE, SURMOD). A program for furrow irrigation (FISDEV) is under development at the Center for Irrigation Engineering in Leuven. Pressurized irrigation programs were few and concentrated on spray patterns and hydraulics (CATCH3D). BASCAD and CATCH3D are easy-to-learn practical tools, most others are meant for experimental analysis.
SUMMARY, RECOMMENDATIONS

The programs on canals and canal networks included a few simple design tools for steady flow based on Manning’s formula (CID, PROFILE, CANAL). Every educational institution seems to develop its own program for this purpose and versatility rather differs. Available mature models which also consider non-steady flow in canal systems seem not to be widely available (CANAL, DUFLOW, MODIS), although more programs have been developed "in house" (CARIMA, LYMPHA, SAMWAT database). DUFLOW is good, but has irrigation only as a sub-set. A French model (SIC) developed in connection with the IIMI is simply too expensive for our target group, as is a Dutch model (MODIS).

Pipes and pipe networks programs appeared to be relatively obscure: very few were available and did not even work (OPTIPIPE). Apparently they either are simple diameter calculations belonging "in house" (HAZEN-W, DRIPSPI), or are not publicly available, such as network solvers used in Israel. A strong connection with the drinking water supply discipline was noted (UNDP), where many networks are simulated or operated using computers (WATNET). An upcoming New Zealand program looks promising for pressurized system designers (IRRICAD).

Under Structures, in fact only one program (FLUME) was found for the analysis and design of broad-crested weirs, with some similar but simpler programs based on the same theory (BCW/RBC; BCWEIR). Design programs used for silt trapping structures were mentioned to be in use at Hydraulics Research in Wallingford.

The Irrigation system management chapter revealed that simple allocation could be done using spreadsheets (MAINSYST), which are also effective for information system management (MIS). The more complex operational and management models (OMIS, INCA) are attractive, but expensive affairs, outside the budget of our target group. The large databases necessary and the interactive communication ask for larger memory, faster access, and generally more modern technology (INCA). A more modest approach has been used in Burma and Thailand (WASAM), but the program requires upgrading, while other attempts are still under development (CAMISIS, CIMIS). There is considerable scope for effective computer use in this area.

The Drainage chapter indicated that there are a few "internal" single purpose programs available (AUGER, DRAINCAN), that some soil moisture based programs have been developed but are not easy to transfer (SWATER, SWACROP), that in the groundwater area some useful tools have been made (SGMP, SATEM), and that salinity programs are mostly still under development (SALTMOD, REUSE). The number of integrated programs seems to be limited (DRAINMOD), and design programs require additional software (DrainCAD).

It was finally mentioned that there are undoubtedly more programs than we have described, especially also in areas more or less closely related with irrigation, such as hydrology and meteorology. Further, a minor irrigation design program under
development was mentioned (MIDAS), as well as a relatively simple planning simulation package from India (SIMYIELD). They did not fit in our simple pragmatic classification chapters.

14.2 General conclusions

We concede that a total of forty-odd test-run programs may be a relatively small sample of potentially useful irrigation software, available worldwide. There is, in addition, a logical bias towards our immediate surroundings: what is available at greater distance is more difficult to see. However, within the limitations of time, money, and knowledge, this is what we could readily obtain. And on the basis of this sample, we shall draw conclusions, which we hope will have a more general dimension. To a certain extent, we find support in the literature (referred to in previous chapters) for this wider applicability of many of our conclusions.

As far as the subject areas are concerned, we can conclude that there are some overlapping efforts in water requirements and irrigation scheduling, where seemingly similar programs are appearing in various places, often based on FAO Irrigation and Drainage Papers 24 and 33. Such an overlap is also apparent in canal models, which has already led to the advice to direct efforts towards improving existing models rather than creating yet another model. In irrigation system management models there is a similar danger visible, although developed software packages are, so far, rather site-specific. A proper co-ordination of efforts apparently does not exist.

Areas which are rather under-represented in the availability of computer programs seem to be pipes and pipe networks and irrigation structures. For irrigation structures there is virtually only one type of weir coming out of our survey. For pipes and pipe networks there are also very few programs mentioned, but the impression was obtained that pipe network computer software for other than irrigation purposes could be useful, and that pressurized irrigation network programs are made in more commercial environments. A more definite conclusion is not possible.

With the usability criteria, as outlined earlier, in the back of our mind, we further see that the test-run programs reveal the following state of affairs:

- **hardware requirements** and the need for additional software are limiting in a few cases only;
- the **user-friendliness** of many programs is deploring, as exemplified by:
  - a considerable number of programs require an extensive and time-consuming study of the manual before the program can be operated;
  - some irrigation programs do not have a proper screen lay-out with menu choices, lack standard guidance like on-screen Help and error messages, use odd key strokes, and are not fool-proof;
  - many of the Fortran programs do not have interactive data input, but require cumbersome and inflexible filling of ready-made data forms;
some programs have no file handling system, leaving the less-experienced computer user with a confusing task;

functional graphics to ease data input and the quick analysis of output is completely absent in many programs;

many older programs apparently are not maintained or updated to make them more friendly for today's user;

- the manual is problematic for quite a number of programs: either absent completely, or poorly written and edited, or lacking basic parts like theoretical background, program logic, installation routines, sample cases;

- the availability of a number of programs also creates difficulties: the purchase price is sometimes too high, in other cases programs are described in the literature but not marketed, or they are only meant for internal use, or constantly under development.

It is also striking that a number of available programs do not work at all or abort during test runs.

Apart from the subject area of irrigation software, one may distinguish various purposes like research, analysis, design, and training and education. It appears that many programs or simulation models bear the imprint of the individual researcher, who has great attention for the conceptual, scientific and theoretical detail. Although this is completely justified, there is much less attention for the subsequent step of preparing a program or a model for wider distribution and marketing. The researcher's hesitations (program under development, needs improvement, etc.) are clearly visible in quite a number of cases and irrigation software centres, with attention for publishing and dissemination (and hence user orientation) are still scarce. Computer-aided design programs for irrigation are relatively few in number, although an increasing number of tools (small programs which assist in a design detail) are available. Most design programs are relatively recent, i.e. dating from the past few years. It further appears that very few programs (except some games) were developed for the sole purpose of education or training, and computer-assisted instruction in irrigation is still in its infancy. Of course, some programs can be used for more than one purpose, but this multi-purpose character is often assumed afterwards rather than intended from the start.

The situation is not completely negative, however, since there are also programs that have adequate attention for the user interface, that have a complete and clear manual, and that are easily available at a reasonable price. It is remarkable, that especially the relatively simple single-purpose programs have a positive score on the above counts and are appropriate for their task. In addition, one sees a group of appropriate, more comprehensive and complex programs arriving on the market, which are based on modern software engineering principles, criteria and techniques, but which are often rather expensive. The "middle of the market" seems to receive relatively less attention.

Overseeing these rather critical remarks, we may conclude that there is a surprisingly small number of irrigation programs that are up to standard, complete, available, quickly to learn, and easy to use for our target group. It looks as if many programs on
the market were made for one specific purpose and then, as an afterthought, made available to other interested parties, without further adjustments. Of course, it could be useful to learn from other attempts, but relatively little seems to have been done in the way of user considerations. In some cases the programs demonstrate little structured programming, while in other cases software engineering is overshooting our target group. A general consensus on the need for quality software production, and how to realize this, seems to be missing.

As far as the production and maintenance of irrigation software is concerned, there also seem to be three or four institutions which are developing and marketing more than the incidental microcomputer program (Logan, Leuven, Wallingford and CEMAGREF). There is a second echelon, where reasonable programs are produced, but in lower numbers and frequency (FAO, Delft Hydraulics, USWCL, ILRI, etc.). Despite critique on some of their programs, the few main centres seem to head in the right direction of placing computer software development for irrigation on a professional and more structural footing. This does not mean that the occasional program originating from other institutions cannot be useful, but there often is a lack of a standardized approach. The "ideal" defined in Chapter 13 is not normally reached in such cases. We hope that more overviews such as this one may assist in more attention for uniformity, usability and other standards in the production and marketing of irrigation application programs.

In the process of developing adequate quality software for practising irrigation engineers, we may conclude that we are apparently still in the early stages of development. Part of this may be due to fast-changing developments in computer technology. But, despite great strides forward in the necessary knowledge over the last five years or so, the great range in quality of currently available programs and the rudimentary institutional build-up for the subject of irrigation software indicates that there is a long way to go before appropriate programs are produced and published as matter of routine.

14.3 Recommendations

Stemming from the foregoing summary and conclusions, and from some more general considerations given earlier in the report, the following suggestions are formulated in connection with irrigation-practitioner oriented microcomputer software:

- More institutions should consider putting some extra effort into applying finishing touches to "internal" programs, written with a lot of effort, so that they are made easily usable by others in the same irrigation field;
- More contact between various institutions dealing with irrigation software seems required: a network or a model centre could help. The existing collaboration between Leuven/Wallingford/ILRI is a step in the right direction;
- Standards and criteria for the production and maintenance of irrigation software should be formulated and adhered to as much as possible. Criteria outlined in this publication could serve as a starting point for development of usability aspects;
SUMMARY, RECOMMENDATIONS

- The development of training/education modules via the CAL/CAI system is an area which should be more energetically explored; applied education science and irrigation could meet to advantage in this respect;
- The collection and preparation of a set of simple, single purpose calculation aids for irrigation should be followed up. The term computational irrigation is relevant, while terms like a "tool box" may be applicable;
- The development of Expert Systems in irrigation for improved decision support should be stimulated;
- The use of GIS and other information systems in irrigation should become more widely discussed and possibilities for its application more widely explored;
- Inventories like the current one should be improved, extended and repeated/upgraded at regular intervals;
- The preparation of a thorough annotated bibliography on the topic of this publication could be a useful addition to the knowledge base;
- The applicability of software from irrigation-related or even further removed disciplines should be better and more fervently explored;
- Training in the application of irrigation computer software should be increased so as to avoid duplication and unnecessary programming work;
- The awareness of the perspectives, possibilities, perils and pitfalls of computer use in irrigation research, design, management and training should be an integral part of professional education and training.

Finally, we would like to repeat our call from the first chapter: any comments, corrections or additional information would be very much welcomed and will be taken into account in any possible future edition.