US Army Corps
of Engineers
Hydrologic Engineering Center

Issues for Applications Developers

January 1993

TP-139

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
January 1993	Technical Paper	

4. TITLE AND SUBTITLE
Issues for Applications Developers

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Darryl W. Davis

5d. PROJECT NUMBER

5e. TASK NUMBER

5F. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Corps of Engineers
Institute for Water Resources
Hydrologic Engineering Center (HEC)
609 Second Street
Davis, CA 95616-4687

8. PERFORMING ORGANIZATION REPORT NUMBER
TP-139

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/ MONITOR'S ACRONYM(S)

11. SPONSOR/ MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Development of the right applications software for the water industry that is robust, flexible, maintainable, and portable requires a strategy that determines user needs, creates software in a develop, test, user feedback process, and includes training and support. Software engineering decisions related to the choice of engineering methodologies, program architecture, coding languages, graphics and other support libraries, and adoption of hardware ad software industry standards are critical to success. Development of engineering applications software is best accomplished by organizations with experience in both the problem addressed and software development and support.

15. SUBJECT TERMS
computer software, computer applications, computer models, water resources, hydrology, hydraulics

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER
U	U	U	UU	18	

Issues for Applications Developers

January 1993

US Army Corps of Engineers
Institute for Water Resources
Hydrologic Engineering Center
609 Second Street
Davis, CA 95616

(530) 756-1104
(530) 756-8250 FAX
www.hec.usace.army.mil

TP-139

Papers in this series have resulted from technical activities of the Hydrologic Engineering Center. Versions of some of these have been published in technical journals or in conference proceedings. The purpose of this series is to make the information available for use in the Center's training program and for distribution with the Corps of Engineers.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products.

Issues for Applications Developers[1]

DARRYL W. DAVIS, BS, MS, Member ASCE
Director, Hydrologic Engineering Center, U.S. Army Corps of Engineers.

SUMMARY

Development of the right applications software for the water industry that is robust, flexible, maintainable, and portable requires a strategy that determines user needs, creates software in a develop, test, user feedback process, and includes training and support. Software engineering decisions related to the choice of engineering methodologies, program architecture, coding languages, graphics and other support libraries, and adoption of hardware and software industry standards are critical to success. Development of engineering applications software is best accomplished by organizations with experience in both the problem addressed and software development and support.

1. INTRODUCTION

Applications software are important tools used by the water resources community for planning, design and operation of water resource projects. Desktop hardware, operating systems, coding languages, and a myriad of other factors have evolved such that the traditional applications development environment of an engineer writing FORTRAN code is no longer appropriate. The software used in the coming decade will be highly sophisticated from a technical standpoint, constructed specifically for the user environment, and include advanced graphical display capabilities.

There are several important questions for the water engineering community to address. What should the software do? How and in which environment should it function? How should this be determined? Who should develop this software? Who (and how) should support the software? How can the profession ensure that user needs will be adequately reflected? The answers to these questions are of interest because a new generation of applications software is under development by governments, academia, and the commercial sector. This paper summarizes "truisms" related to engineering software development and technology transfer and offers commentary related to these questions.

2. A PROVEN SOFTWARE DEVELOPMENT STRATEGY

A method for successfully accomplishing software development, implementation and servicing is as follows: a) need for new methods and procedures surface through solving real-world problems and maintaining contacts with the user community, b) research and development work is performed to solve specific problems, c) solutions are generalized so that they may service other problems, d) high quality documentation is developed and software is prepared for long term service and maintenance, e) training courses are held and consultation projects performed that gradually, but systematically, move the software into every day work of users, and f) continuing development, servicing and maintenance are performed to assure aid to users and guarantee up-to-date capabilities are incorporated.

---

[1] Invited Keynote Presentation at WATERCOMP '93, 2nd Australasian Conference on Computing for the Water Industry - Today and Tomorrow, 30 March - 1 April 1993, Melbourne, Australia.

2.1 Observations for Applications Package Developers

Several "truisms" have emerged that are applicable to the development and implementation of engineering applications packages. These observations are directed to a unit in an institution (public or private) that is developing new applications software and provides service and support to in-house and other users.

a) Large scale, complex, comprehensive computer programs are dynamic entities that require continuous nurturing and support in order to remain viable and useful. Such computer software needs a permanent home; an institution that is philosophically committed to the improvement in procedures, morally committed to servicing and improving the programs, competently staffed to perform that task, and available "on call" to users.

b) Professionally developed computer program code and its management is vital for software to be effectively maintained and be portable among hardware platforms. Use of special purpose languages that are proprietary or are not generally within platform and software industry standards should be avoided. Adherence to "standards" such as American National Standards Institute (ANSI) language standards is important and use of modern programming practice is needed to minimize difficulties in computer source code maintenance.

c) Successful implementation of advanced applications packages requires both useful technology available in appropriate form and users that are interested and anxious to take advantage of the opportunities. It is important in early stages to encourage applications that are manageable and have potential for success. A commitment to a service attitude and genuine interest in solving user community specific problems are basic.

A series of do's and do not's with supporting explanation follows which attempts do define a framework and strategy for applications software development and implementation.

a) Engineering management should not "require" applications packages to be used before considerable experience and shakedown is accomplished. Nothing kills interest in a new package like forced use that does not deliver the solution to everyone's problems. New applications packages cannot be so tightly developed that they can survive an environment wherein the potential users are put in a negative posture by the forced approach. Pragmatic, steady, gradual introductions will likely result in early, meaningful use of the concepts and techniques. Nothing draws users like success, no matter how small.

b) Avoid (if possible) the grand "demonstration" exercise. Application demonstrations designed to sell technology often get too many people involved with parochial agendas. The exercise often becomes rigged or fails because of the weight of so many observers. Dissemination of basic information through publicizing applications is useful. Including sessions on the application in seminars, general meetings, and training courses is an excellent method for exposing applications packages to potential users.

c) Work with users to solve their problems. A full commitment to solving the users problem is perhaps the single most important facet of successful technology transfer. An approach that solves specific problems from which the elements are continuously merged into an analytical system is more responsive to user needs than creating a grand solution that is then adapted to a specific problem. It is not unusual for an application to have some unique twist. Early implementation efforts should seek to work with users on specific studies.

d) Carefully select manageable studies or portions of studies for initial applications. This is the operational implementation of the idea that nothing draws users like success, no matter how small. The selection of small well-defined problems that both developers and users can learn from and thus improve the program is important. A poor strategy attempts to "solve the unsolvable" as an early application. There are always difficult problems needing solution; build an experience base before stretching too far. A series of small, growing to more comprehensive and difficult applications over time is the desired strategy.

e) Be prepared and willing to perform logic and program code changes for early studies. Developers usually cannot foresee all potential study environments, objectives, data availability, issues, etc. for which the software might be used. Design deficiencies, bugs, and errors will exist. The attitude and ready resources to make the necessary adjustments will reflect the commitment to a services approach to implementation.

2.2 Observations for Applications Package Users

The successful user is one that is confronted with a problem, has struggled to find a solution, and recognizes that it could be at least partially addressed with the applications package. The unsuccessful user is often the recipient of an applications package provided by a colleague or superior. The colleague or superior was probably introduced to the software in a general way and became convinced that it must surely have value, especially if appropriately used by others, (the user) to solve his problems. With these positions defined, a few comments are offered below.

a) Know problems and needs in detail. There is a tendency for users, especially those who are not highly computer oriented, to end up with their problems becoming defined by the performance capabilities of a particular software package. This results in a reverse approach to acquiring a high technology solution to a problem and is usually not the best approach.

b) Determine how the problem should be solved irrespective of the capabilities of applications packages. Sophisticated applications packages require considerable commitment of resources, both dollars and manpower. The potential user should make certain that resources are effectively used to accomplish the problem solution that generated the search for the applications package.

c) Thoroughly investigate features and capabilities of alternative applications packages. Applications packages come in integrated hardware-software arrangements, software alone, or just specific-task oriented software. Important issues are propriety of the package (Is a license required and what are the costs and restrictions?), specialized nature of hardware platforms and peripherals, software package adherence to standards, documentation, service, training, and compatibility with existing and future equipment and people. What is right for one circumstance may not be relevant to another.

d) Do not expect magic. Applications packages performance between hardware and system environments can vary greatly. While one should prudently seek a package that has a record of minimum difficulties, it is best to plan for at least some start-up time and remain flexible. Start-up should be well planned and involve user representatives.

e) Willingly commit the personnel resources to "own" the applications package. A major shortcoming in the effective use of sophisticated applications packages is the unwillingness of potential users to devote adequate time and energy to "own" the software package in an applications sense. Most capable engineering applications packages are sufficiently sophisticated that continuous use and familiarity by the users is needed to maintain effectiveness.

f) Continuously ask questions of the developers and user supporters. Probe the limits of capabilities, and presume sophisticated software should be continually adapted and improved over time. A package frozen in capability from installation date is one that will soon be unresponsive to the needs of the users. When evaluating and using engineering products, it is of primary importance that the user truly understands the product. A first-rate engineer that truly knows what he is doing will likely produce a better solution using a second-rate applications product, than a second-rate engineer could do using a first-rate applications product he doesn't understand well.

3. DEVELOPING THE RIGHT APPLICATIONS PACKAGE

Determining user needs is the critical first step in the development of a successful applications package. Software engineering, a discipline that addresses the complete software development cycle, continues to propose, test, and refine strategies for ensuring successful software development projects. A popular software engineering approach often referred to as the "waterfall model" includes performing the following: requirements analysis, preliminary and final design, coding, testing, deployment, and service and support. The process is conceived of as once through, beginning to end, permitting an efficient, manageable, production oriented approach. Users define the needs and software specialists design, code, test, and deploy the product. Some interaction with users is anticipated during the development process.

Experience suggests that while this approach is a useful framework, development of successful engineering applications software is best served by a less formally structured, multi-pass approach. The organization and its staff that is assigned the development project is important. Organizations and staff that have experience in performing studies in the technical area of interest, developing and deploying applications packages, and training and support are best suited to performing the work.

The requirements analysis step is useful and essential. Preliminary requirements are defined by a development team in consultation with a selected group of user representatives. The preliminary requirements are documented and circulated among a larger user group for comments and input. The developers in consultation with the selected group of users then prepare final requirements. Development of a prototype (or limited-scope preliminary version) can be very helpful at this stage by providing a real, functioning program (as compared to a paper plan) to which potential users may respond. A certain amount of design will have taken place during the prototype development. Its best to take time to perform a complete conceptual design that will be tested in the prototype development. Flexibility for future improvements key.

Development of the applications package can then be undertaken as a production process. For a sophisticated and capable engineering applications package, the development team should be comprised of a specialist in the technical applications area (often the team leader), and a complement of computer scientists, programmers, and consultants. In today's technology environment, the development of an engineering applications product requires the combined talents of knowledgeable engineer-practitioners and skilled computer science specialists. It is no longer possible for a few engineers to possess the broad range of skills necessary to produce a satisfactory product. Not all team members need to be full-time on the project. The consultants may be from other groups in the organization or procured via contract to provide limited scope, highly specialized knowledge that is essential in the extremely capability-rich yet complex hardware and systems environment.

Development should be staged so that usable products emerge in a regular manner throughout the development period. Product releases should be often enough to provide the user community with the opportunity to observe progress and provide feedback on needed capabilities, but not so often as to create a climate of turmoil and distraction for the developers. Six-month intervals is probably too short with

one-year intervals about right. The first release after the prototype should be a preliminary yet fully functional package. Early releases should be to selected users that are willing to apply the package to real problems but who are familiar with software development so that difficulties that will arise are not unexpected.

4. HARDWARE, OPERATING SYSTEM, CODING, AND RELATED STANDARDS

Today the typical engineering computing environment has become the desktop machine. It is likely to be a high-end personal computer with an Intel 486 processor (soon to be succeeded by P5), or a RISC-chip based engineering workstation (or X-Terminal to a workstation) equipped with a high-resolution color monitor. The desktop machine is connected to other workstations, file servers, laser printers, plotters and other devices via a local area network. In some instances, access to regional centers and other national and international sites is available through network gateways to worldwide communication facilities.

The software developer must design and develop applications packages to take advantage of the opportunities provided by this rich environment. Developers must be careful to avoid constructing applications that exhibit hardware and system dependencies that adversely affect code portability, future upgrades, and long-term servicing. Most software industry professionals and users generally agree that these notions are highly desirable; the goals are easy to articulate. The pay-off is in the successful translation into software development strategies, standards, criteria, and ultimately computer code that achieves those goals.

4.1 Hardware/operating System

Hardware and associated "chip" families, operating systems, and binary (compiled and linked) code compatibility are tightly connected. For example, MS-DOS [1] and Microsoft Windows [2] operate within the Intel-chip family of personal computers and thus binary code is compatible among machines. There are a variety of RISC chips that are used in workstations. Binary code is generally compatible within a chip family (vendor product line); for example among the IBM RISC-chip workstation line of computers, but not across chip/vendor computers. UNIX [3] is the standard operating system for RISC-chip engineering workstations providing code compatibility at the source (not binary) level. While this is not particularly important for the user, it is extremely important for program developers.

Minimum hardware configuration and specifications that can be expected for the engineering desktop for the next few years are as follows:

> Personal Computer: Intel 486/66 mhz processor, 8 to 32 MB RAM, 200 to 600 MB disk, 14" Super VGA monitor, networked to plotters and printers, DOS/Windows operating environment.
>
> Workstation: RISC-chip/50 mips processor, 32 MB RAM, 1 gigabyte disk, 17" monitor, networked to other workstations and peripherals locally and regionally, UNIX operating system.

For the personal computer, DOS has been the unquestioned standard for office automation applications. While there are a number of capable engineering applications packages running in DOS, the future seems to be toward multi-tasking, window-based systems. Candidates are Microsoft Windows (soon to be Windows NT [4]), OS/2 Presentation Manager [5], and UNIX. In the RISC-chip based workstation environment, UNIX is the standard, with the possibility that Windows NT might soon be a competitor for some chip families. It is important to maintain adherence to a standard, such as Posix [6] to ensure cross UNIX platform compatibility.

For the software developer, the issue is therefore what hardware configuration, likely operating systems, coding languages and associated compilers, third-party libraries, etc. will enable the desired performance, portability, upward compatibility, and service support needed for the applications package. The likelihood is that packages will need to be functional in both environments. The appropriate strategy to follow is to code the application using languages, libraries, utilities etc. that make it least painful to port to other platforms. This is easier said than done.

4.2 Programming Philosophy, Languages, and Related Issues

The application package to be developed must ultimately be coded in a computer language, compiled, and linked into binary code for execution on a specific platform. Various programming strategies, languages, and use of commercial utilities and libraries are employed. Historically, an engineer programming in FORTRAN and following the ANSI language standard developed engineering applications programs. Often the program in current use was originally coded in FORTRAN II, with subsequent improvements coded in FORTRAN IV, 66, and 77 and ported and re-compiled for the successor generation platforms. This continued to be successful and relatively simple while programs read mostly number and character input and output the same.

The base engineering functions that implement the solution algorithms are becoming more and more transportable across a wide variety of chip families. This is true whether they are coded in FORTRAN, C [7], or another popular language. Data base access, graphical user interfaces (GUI), and visualization tend to inhibit transportability across platforms at the current time. New languages have emerged responsive to the needs, and an impressive array of commercial libraries and higher level coding aides are available to be used by the programmer. While no definitive consensus has emerged, there are a number of logical strategies to consider in programming the applications package.

The graphical user interface is the boon and the bane of the programmer. It offers the opportunity to create a comfortable and highly productive user environment. The developer must be careful, however, to avoid dressing up a poor or outdated engineering solution with an attractive user interface. The engineering algorithms must be top-notch in order to warrant the considerable effort to create a productive GUI.

Most recently developed GUI are coded in C using standard Motif [8] and X Windows [9] library functions because of the platform portability and power in providing direct programmer control of the user-device interface. A programming concept referred to as object-oriented programming (OOP) [10] is emerging as an important player in the user interface, as well as other, programming areas. It's reported power is that of enabling the creation and manipulation of reusable coded objects that can substantially improve the robustness and maintainability of the software and productivity of the programmer. The coding language that is gaining a following for implementation of OOP is C++ [11]. A number of major commercial software vendors are reported to have adopted C++ for their own new program development. Motif and Open Look [12] provide widget libraries that prescribe a standard look and feel for constructing GUI's in the X Window system. X Windows is the de facto standard windowing system for the UNIX operating system. In the DOS environment, Microsoft Windows is dominant with IBM OS/2 Presentation Manager also a player.

Unfortunately, a GUI developed following standards in the UNIX workstation environment is not directly portable to Microsoft Windows, and vice versa. Since engineering applications packages will most likely need to function in both systems, a dilemma exists. One approach is to develop separate GUI's for each environment. While unattractive, its done in the commercial sector. Another is to use proprietary GUI builder libraries and cross platform compilers. This is also unattractive, perhaps even

more so. The best approach seems to be to proceed with development following the prevailing standard in each (say Motif and Microsoft Windows), isolate the code related to the GUI from other program functions, and take care to be as consistent between both environments as possible. One also hopes that the next few years continue the trend toward a common operating system and attendant GUI standards that will serve both environments.

The majority (perhaps above 90%) of currently used engineering applications program "engines", the engineering algorithm solution portion of the program, are coded in FORTRAN. This is likely to continue for some time for new programs as well. This is both because engineering programs tend to be developed by engineers, and routines from the substantial inventory of functioning FORTRAN programs will be re-used in new programs. FORTRAN 90 [13], the next ANSI FORTRAN standard, offers new data structures, dynamic memory, and other desirable attributes. Some industry observers have suggested that future FORTRAN standards and extensions will implement OOP concepts more fully. A number of software development projects [14], are being developed with OOP concepts using C++ for the overall program architecture, C where necessary, and FORTRAN for some compute functions.

4.3 Graphics

Increasing use of display and output graphics (often referred to as visualization) is the emphasis for the future for engineering applications packages. Coding the graphics routines using primitive, basic level intrinsic from libraries may be logical and practical for mass-market commercial software firms. It is not often practical for the more limited market of engineering applications programs. Making calls from the applications program to graphics functions routines is more common. The question then is which package of graphic function routines should be used? Again, the circumstance is complicated so the best choice is not obvious.

The choices reduce to selecting from commercial and public domain packages (there are quite a number) such as UNIRAS [15] and InterViews [16] that provide graphics products on the fly from simple program level calls. Decision factors include capability, licensing and fee arrangements, documentation and support, platform availability, and success history in the market place. All things being equal, one would select the package that has adequate capability, is in the public domain thus minimizing licensing and fee issues, is available for target workstation and personal computer platforms, and is reasonably documented and supported.

No package has emerged that has gained significant market acceptance that supports both workstation and personal computer platforms. If the application will be run only in the Microsoft Windows environment and the Windows graphics library is adequate, it is an attractive choice. This is not often the case but in the near term, it may be a reasonable alternative for the personal computer implementation of the applications package. The use of X Windows libraries provides such capabilities in the UNIX environment. No clearly dominant commercial or public domain high-level graphics support package has emerged for programming applications for either Microsoft windows or X Windows workstations. It is desirable that products be developed such that they may interface to still-higher level graphics capabilities available in geographic information systems packages.

4.4 Data Base Support

An important issue for software development projects is providing for data persistence necessary to support the GUI, graphics, and technical analysis envisioned. Depending on the applications package, many data types must be addressed. These could include time-series, (hydrologic data), paired-function (x, y tabulations), model-parameter, stream-geometry, and spatial and image data. Data base management

systems were created to meet such needs. The larger, more complex in scope the applications package, the more likely that significant amounts of data of several types might be important. No single data base system, commercial or private, seems to offer efficient management for the full range of data types. Commercial systems, for example ORACLE [17], offer great capability for managing relational data, but limited capability for time series data. Specialized systems, for example HEC-DSS [18], are optimized for time-series and paired-function data.

Developers should carefully analyze the data management needs for their specific applications package, and design early, the approach to be taken. The increasing availability of industry-wide and regional databases that may be useful for application packages warrants consideration in program design. Also, the need to share (or pass to the next step in design), engineering data is an issue that should be considered as well. Whether to design a custom-coding solution, or choose from commercial and public domain data base packages is a decision that should consider portability, license and run-time fees, programmer effort to implement, and requirements for long term service and support.

5. CONCLUSIONS

Successful development of the right engineering applications software packages requires adopting a strategy that determines user needs, and accomplishes development in a develop, test, user feedback process. Application package development should be performed by organizations that have experience in solving engineering problems in the field, experience in developing, deploying, maintaining and supporting applications software, and are committed to a services approach to users. The development team should be comprised of a technical specialist in the applications area, and a complement of computer scientists and programmers. The engineering desktop platforms for the next few years include high-end Intel-chip personal computers and RISC-chip based workstations. Use of modern software architecture concepts to include OOP, application of standard programming languages, and adherence to published software standards (where they exist) and de-facto industry standards are essential to ensure successful applications package development.

6. ACKNOWLEDGEMENTS

The views expressed in this paper are a synthesis of the experience of the staff of the Hydrologic Engineering Center. This experience was gained from 25 years of developing and supporting engineering applications software for the U.S Army Corps of Engineers.

7. REFERENCES

1. Microsoft Corporation, "MS-DOS User's Guide and Reference Version 5.0", Microsoft Corporation, 1991.
2. Microsoft Corporation, "Microsoft Windows Version 3.1 User's Guide", Microsoft Corporation, 1992.
3. Rosen, Kenneth H., Rosinski, Richard, R., and Farber, James M., "UNIX System V Release 4: An Introduction", Osborne McGraw-Hill, 1990.
4. Microsoft Corporation, " Microsoft WIN32 SDK for Windows NT (Preliminary)", Microsoft Corporation, 1992.
5. IBM Corp., "Operating System/2 Standard Edition User's Reference", IBM Corp., 1987.
6. Levine, Donald A., "Posix Programmer's Guide", O'Reilly & Associates Inc., 1991.

7. Kernigham, Brian W., Ritchie, Dennis M., "The C Programming Language", Prentice Hall, 1988.
8. Open Software Foundation, "OSF/Motif Programmer's Reference", Prentice Hall, 1991.
9. Asente, Paul, and Swich, Ralph, " The X Window System Toolkit", DEC Press, 1990.
10. Cox, B., "Object-Oriented Programming: An Evolutionary Approach", Addison-Wesley, 1986.
11. Ellis, Margaret A., and Stroustrup, Bjarne, "The Annotated C++ Reference Manual", Addison-Wesley, 1990.
12. Sun Microsystems, Inc., "OPEN LOOK Graphical User Interface Functional Specification", Sun Microsystems, Inc., 1989.
13. Microsoft Corporation, "Microsoft FORTRAN Version 5.1 Reference Guide", Microsoft Corporation, 1992.
14. Davis, Darryl W., "The HEC NexGen Software Development Project", Proceedings of Watercomp93, The Institution of Engineers, Australia, 1993.
15. UNIRAS A/S, "agX/Toolmaster Reference Manual", UNIRAS A/S, 1991.
16. Linton, Mark A., Vlissides, John M., and Calder, Paul R., "Composing User Interfaces with InterViews", IEEE Computer, Vol. 22, No.2, 1989.
17. Oracle Corporation, "Professional ORACLE 5.1 A Reference Manual", Oracle Corporation, 1988.
18. USACE Hydrologic Engineering Center, "HEC-DSS User's guide and Utility Program Manuals", 1990.

Technical Paper Series

TP-1 Use of Interrelated Records to Simulate Streamflow
TP-2 Optimization Techniques for Hydrologic Engineering
TP-3 Methods of Determination of Safe Yield and Compensation Water from Storage Reservoirs
TP-4 Functional Evaluation of a Water Resources System
TP-5 Streamflow Synthesis for Ungaged Rivers
TP-6 Simulation of Daily Streamflow
TP-7 Pilot Study for Storage Requirements for Low Flow Augmentation
TP-8 Worth of Streamflow Data for Project Design - A Pilot Study
TP-9 Economic Evaluation of Reservoir System Accomplishments
TP-10 Hydrologic Simulation in Water-Yield Analysis
TP-11 Survey of Programs for Water Surface Profiles
TP-12 Hypothetical Flood Computation for a Stream System
TP-13 Maximum Utilization of Scarce Data in Hydrologic Design
TP-14 Techniques for Evaluating Long-Tem Reservoir Yields
TP-15 Hydrostatistics - Principles of Application
TP-16 A Hydrologic Water Resource System Modeling Techniques
TP-17 Hydrologic Engineering Techniques for Regional Water Resources Planning
TP-18 Estimating Monthly Streamflows Within a Region
TP-19 Suspended Sediment Discharge in Streams
TP-20 Computer Determination of Flow Through Bridges
TP-21 An Approach to Reservoir Temperature Analysis
TP-22 A Finite Difference Methods of Analyzing Liquid Flow in Variably Saturated Porous Media
TP-23 Uses of Simulation in River Basin Planning
TP-24 Hydroelectric Power Analysis in Reservoir Systems
TP-25 Status of Water Resource System Analysis
TP-26 System Relationships for Panama Canal Water Supply
TP-27 System Analysis of the Panama Canal Water Supply
TP-28 Digital Simulation of an Existing Water Resources System
TP-29 Computer Application in Continuing Education
TP-30 Drought Severity and Water Supply Dependability
TP-31 Development of System Operation Rules for an Existing System by Simulation
TP-32 Alternative Approaches to Water Resources System Simulation
TP-33 System Simulation of Integrated Use of Hydroelectric and Thermal Power Generation
TP-34 Optimizing flood Control Allocation for a Multipurpose Reservoir
TP-35 Computer Models for Rainfall-Runoff and River Hydraulic Analysis
TP-36 Evaluation of Drought Effects at Lake Atitlan
TP-37 Downstream Effects of the Levee Overtopping at Wilkes-Barre, PA, During Tropical Storm Agnes
TP-38 Water Quality Evaluation of Aquatic Systems
TP-39 A Method for Analyzing Effects of Dam Failures in Design Studies
TP-40 Storm Drainage and Urban Region Flood Control Planning
TP-41 HEC-5C, A Simulation Model for System Formulation and Evaluation
TP-42 Optimal Sizing of Urban Flood Control Systems
TP-43 Hydrologic and Economic Simulation of Flood Control Aspects of Water Resources Systems
TP-44 Sizing Flood Control Reservoir Systems by System Analysis
TP-45 Techniques for Real-Time Operation of Flood Control Reservoirs in the Merrimack River Basin
TP-46 Spatial Data Analysis of Nonstructural Measures
TP-47 Comprehensive Flood Plain Studies Using Spatial Data Management Techniques
TP-48 Direct Runoff Hydrograph Parameters Versus Urbanization
TP-49 Experience of HEC in Disseminating Information on Hydrological Models
TP-50 Effects of Dam Removal: An Approach to Sedimentation
TP-51 Design of Flood Control Improvements by Systems Analysis: A Case Study
TP-52 Potential Use of Digital Computer Ground Water Models
TP-53 Development of Generalized Free Surface Flow Models Using Finite Element Techniques
TP-54 Adjustment of Peak Discharge Rates for Urbanization
TP-55 The Development and Servicing of Spatial Data Management Techniques in the Corps of Engineers
TP-56 Experiences of the Hydrologic Engineering Center in Maintaining Widely Used Hydrologic and Water Resource Computer Models
TP-57 Flood Damage Assessments Using Spatial Data Management Techniques
TP-58 A Model for Evaluating Runoff-Quality in Metropolitan Master Planning
TP-59 Testing of Several Runoff Models on an Urban Watershed
TP-60 Operational Simulation of a Reservoir System with Pumped Storage
TP-61 Technical Factors in Small Hydropower Planning
TP-62 Flood Hydrograph and Peak Flow Frequency Analysis
TP-63 HEC Contribution to Reservoir System Operation
TP-64 Determining Peak-Discharge Frequencies in an Urbanizing Watershed: A Case Study
TP-65 Feasibility Analysis in Small Hydropower Planning
TP-66 Reservoir Storage Determination by Computer Simulation of Flood Control and Conservation Systems
TP-67 Hydrologic Land Use Classification Using LANDSAT
TP-68 Interactive Nonstructural Flood-Control Planning
TP-69 Critical Water Surface by Minimum Specific Energy Using the Parabolic Method

TP-70 Corps of Engineers Experience with Automatic Calibration of a Precipitation-Runoff Model
TP-71 Determination of Land Use from Satellite Imagery for Input to Hydrologic Models
TP-72 Application of the Finite Element Method to Vertically Stratified Hydrodynamic Flow and Water Quality
TP-73 Flood Mitigation Planning Using HEC-SAM
TP-74 Hydrographs by Single Linear Reservoir Model
TP-75 HEC Activities in Reservoir Analysis
TP-76 Institutional Support of Water Resource Models
TP-77 Investigation of Soil Conservation Service Urban Hydrology Techniques
TP-78 Potential for Increasing the Output of Existing Hydroelectric Plants
TP-79 Potential Energy and Capacity Gains from Flood Control Storage Reallocation at Existing U.S. Hydropower Reservoirs
TP-80 Use of Non-Sequential Techniques in the Analysis of Power Potential at Storage Projects
TP-81 Data Management Systems of Water Resources Planning
TP-82 The New HEC-1 Flood Hydrograph Package
TP-83 River and Reservoir Systems Water Quality Modeling Capability
TP-84 Generalized Real-Time Flood Control System Model
TP-85 Operation Policy Analysis: Sam Rayburn Reservoir
TP-86 Training the Practitioner: The Hydrologic Engineering Center Program
TP-87 Documentation Needs for Water Resources Models
TP-88 Reservoir System Regulation for Water Quality Control
TP-89 A Software System to Aid in Making Real-Time Water Control Decisions
TP-90 Calibration, Verification and Application of a Two-Dimensional Flow Model
TP-91 HEC Software Development and Support
TP-92 Hydrologic Engineering Center Planning Models
TP-93 Flood Routing Through a Flat, Complex Flood Plain Using a One-Dimensional Unsteady Flow Computer Program
TP-94 Dredged-Material Disposal Management Model
TP-95 Infiltration and Soil Moisture Redistribution in HEC-1
TP-96 The Hydrologic Engineering Center Experience in Nonstructural Planning
TP-97 Prediction of the Effects of a Flood Control Project on a Meandering Stream
TP-98 Evolution in Computer Programs Causes Evolution in Training Needs: The Hydrologic Engineering Center Experience
TP-99 Reservoir System Analysis for Water Quality
TP-100 Probable Maximum Flood Estimation - Eastern United States
TP-101 Use of Computer Program HEC-5 for Water Supply Analysis
TP-102 Role of Calibration in the Application of HEC-6
TP-103 Engineering and Economic Considerations in Formulating
TP-104 Modeling Water Resources Systems for Water Quality
TP-105 Use of a Two-Dimensional Flow Model to Quantify Aquatic Habitat
TP-106 Flood-Runoff Forecasting with HEC-1F
TP-107 Dredged-Material Disposal System Capacity Expansion
TP-108 Role of Small Computers in Two-Dimensional Flow Modeling
TP-109 One-Dimensional Model for Mud Flows
TP-110 Subdivision Froude Number
TP-111 HEC-5Q: System Water Quality Modeling
TP-112 New Developments in HEC Programs for Flood Control
TP-113 Modeling and Managing Water Resource Systems for Water Quality
TP-114 Accuracy of Computer Water Surface Profiles - Executive Summary
TP-115 Application of Spatial-Data Management Techniques in Corps Planning
TP-116 The HEC's Activities in Watershed Modeling
TP-117 HEC-1 and HEC-2 Applications on the Microcomputer
TP-118 Real-Time Snow Simulation Model for the Monongahela River Basin
TP-119 Multi-Purpose, Multi-Reservoir Simulation on a PC
TP-120 Technology Transfer of Corps' Hydrologic Models
TP-121 Development, Calibration and Application of Runoff Forecasting Models for the Allegheny River Basin
TP-122 The Estimation of Rainfall for Flood Forecasting Using Radar and Rain Gage Data
TP-123 Developing and Managing a Comprehensive Reservoir Analysis Model
TP-124 Review of U.S. Army corps of Engineering Involvement With Alluvial Fan Flooding Problems
TP-125 An Integrated Software Package for Flood Damage Analysis
TP-126 The Value and Depreciation of Existing Facilities: The Case of Reservoirs
TP-127 Floodplain-Management Plan Enumeration
TP-128 Two-Dimensional Floodplain Modeling
TP-129 Status and New Capabilities of Computer Program HEC-6: "Scour and Deposition in Rivers and Reservoirs"
TP-130 Estimating Sediment Delivery and Yield on Alluvial Fans
TP-131 Hydrologic Aspects of Flood Warning - Preparedness Programs
TP-132 Twenty-five Years of Developing, Distributing, and Supporting Hydrologic Engineering Computer Programs
TP-133 Predicting Deposition Patterns in Small Basins
TP-134 Annual Extreme Lake Elevations by Total Probability Theorem
TP-135 A Muskingum-Cunge Channel Flow Routing Method for Drainage Networks
TP-136 Prescriptive Reservoir System Analysis Model - Missouri River System Application
TP-137 A Generalized Simulation Model for Reservoir System Analysis
TP-138 The HEC NexGen Software Development Project
TP-139 Issues for Applications Developers
TP-140 HEC-2 Water Surface Profiles Program
TP-141 HEC Models for Urban Hydrologic Analysis

TP-142 Systems Analysis Applications at the Hydrologic Engineering Center
TP-143 Runoff Prediction Uncertainty for Ungauged Agricultural Watersheds
TP-144 Review of GIS Applications in Hydrologic Modeling
TP-145 Application of Rainfall-Runoff Simulation for Flood Forecasting
TP-146 Application of the HEC Prescriptive Reservoir Model in the Columbia River Systems
TP-147 HEC River Analysis System (HEC-RAS)
TP-148 HEC-6: Reservoir Sediment Control Applications
TP-149 The Hydrologic Modeling System (HEC-HMS): Design and Development Issues
TP-150 The HEC Hydrologic Modeling System
TP-151 Bridge Hydraulic Analysis with HEC-RAS
TP-152 Use of Land Surface Erosion Techniques with Stream Channel Sediment Models
TP-153 Risk-Based Analysis for Corps Flood Project Studies - A Status Report
TP-154 Modeling Water-Resource Systems for Water Quality Management
TP-155 Runoff simulation Using Radar Rainfall Data
TP-156 Status of HEC Next Generation Software Development
TP-157 Unsteady Flow Model for Forecasting Missouri and Mississippi Rivers
TP-158 Corps Water Management System (CWMS)
TP-159 Some History and Hydrology of the Panama Canal
TP-160 Application of Risk-Based Analysis to Planning Reservoir and Levee Flood Damage Reduction Systems
TP-161 Corps Water Management System - Capabilities and Implementation Status