J. M. Bishop
8/7/84

Quaestiones Informaticae

Vol. 3 No. 2 July, 1984

Quaestiones Informaticae

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Editor: Prof G. Wiechers
Department of Computer Science and Information Systems
University of South Africa
P.O. Box 392, Pretoria 0001

Subscriptions

Annual subscriptions are as follows:

	SA	US	UK
Individuals	R6	$7	£3,0
Institutions	R12	$14	£6,0

Circulation manager
Mr E Anderssen
Department of Computer Science
Rand Afrikaanse Universiteit
P O Box 524
Johannesburg 2000
Tel.: (011) 726-5000

Quaestiones Informaticae is prepared for publication by Thomson Publications South Africa (Pty) Ltd for the Computer Society of South Africa and the South African Institute of Computer Scientists.

Editorial Note

Regrettably this is the last issue of *Questiones Informatica* to appear in its current format. There are a number of reasons for terminating the production of *QI* in printed form. Firstly, the cost of publication has tripled over the last couple of years. Secondly, the interest in the journal has dwindled to the point where one may question the need for a South Arican publication on the more academic aspects of computing: the readership has not expanded but, more seriously, despite many appeals no contributions are submitted. At present I have only one paper in the pipeline. It appears that the only supply of papers comes from the two-yearly Computer Science Symposium, and this is not sufficient to justify the expense of publishing *QI* in its present form.

Nevertheless, both the Computer Society and the Institute of Computer Scientists feel that a vehicle for publishing the results of research in the field of information technology is required for this country. We propose to continue *QI* in a format similar to that used by *Questions Informatica*. In other words we shall use a photo-reproduction process and a printed cover to continue the publication of *QI*, but at greatly reduced costs. This puts the onus of delivering reproducible copy of the authors, but relieves the printers from the problems experienced with unusual symbols, diagrams, computer printouts, etc.

The journal will continue to accept only papers which have been referred. In fact, we shall try to satisfy all the requirements for being regarded as a publication acceptable under the rules for obtaining university subsidies. It is also intended to guarantee quick publication, say not more than three months after acceptance of a paper. However, the journal can only continue to exist if it receives enough contributions of sufficiently high standard.

Again, I would like to appeal to all engaged in research and development in computer related areas, to consider publishing your results in *QI*.

Finally, it is my pleasure to thank Dick White, of Thomson Publications, for all he has done to produce this journal in its current form.

G WIECHERS,
Editor

Software Configuration Management — A practical approach

L.S. du Preez
Software Management Systems

Abstract
A brief description of software configuration management principles is given, followed by a discussion on the application of these principles in a specific installation.

1.0 Overview

As summed up by Bersoff, Henderson and Siegel [1], the software industry in the past has turned out products that have:
- Not satisfied the requirements
- Been delivered much later than scheduled
- Cost more than anticipated
- Been poorly designed

With the ever increasing use of computers, this state of affairs cannot continue. Therefore, software configuration management as a discipline has evolved from the need to make the process of software development more visible to management, in order to apply standard management principles and practices.

For the purpose of this paper, software configuration management is regarded as a discipline imposed by general management external to the software project team. Therefore, only those interactions with the development environment which have an influence on quality assurance are discussed.

This view of software configuration management ensures a clearer distinction between high-level management, such as needed in a military environment [4] where several sub-contractors may be used, and lower level management such as practiced by the project manager. The configuration problems of the latter can usually be resolved by the introduction of software and design standards, adequate backup procedures and work procedures based upon the requirements of high-level management such as those discussed in this paper.

Software configuration management can also be regarded as a quality assurance tool [2], interacting with other quality assurance disciplines such as quality control, verification and validation, and testing and evaluation, as the software life cycle progresses. Because of this, a configuration management system should make provision for checks and balances applicable to the development team, general management and the end user as well.

In the description of software configuration management principles [1] below, four configuration management functions are identified, namely configuration identification, control auditing and status accounting. However, it should be kept in mind that in any given environment these functions form parts of a whole.

The practical application of software configuration principles discussed below, has been used with success in the management of large software projects.

2.0 Configuration Management Principles

2.1 Managing the system life cycle

2.1.1 Software product integrity —

During the life cycle of a software project, user requirements and design concepts are gradually transformed into a set of machine-executable instructions. Therefore, various representations of the components of the software system exist as the life cycle progresses.

The documentation of objectives, specifications and design concepts is therefore a part of the software system as well as program language (i.e. module) listings. These represent the visible form of the software and must always be kept up to date, so that an accurate picture of the invisible software, the executable machine-language statements, can be obtained.

As described in [1], configuration management as a discipline attempts to focus on both the visible and invisible aspects of software representations, with the aim of ensuring product integrity, that is:
- ensuring that user requirements are met
- ensuring the traceability of product transformations during the life cycle
- ensuring that specified performance criteria are met

The realisation of cost and delivery expectations is of course also an integral part of ensuring project integrity. Therefore, product integrity can only be achieved if it is made an objective at the start of a software project so that management effort and resources can be properly allocated. Therefore, software configuration management will only be effective in an environment where project planning and budgeting principles already apply.

2.1.2 Baseline definition

Managing software systems is complicated by the tendency of any system to change, irrespective of its development stage. The further along in the life cycle a change occurs, the more far-reaching will be the consequences of the change.

To facilitate management during the system life cycle, reference points or baselines [1] are formally defined at the end of each stage. If, for example, the following stages in the life cycle of a project are identified, the completion of each will form a baseline for the next stage:
- Definition (Functional baseline)
- Specification (Allocated baseline)
- Development (Product baseline)
- Installation (Operational baseline)
- Maintenance

For a small software project, it may be more convenient to define the first formal baseline at the end of the development stage. On the other hand, when development is based upon a specific contract, the end of the definition or specification stage must form a formal baseline as well. Definition of baselines is a matter of management preference, but software components must be formally updated in a series of steps.

No design baseline is included, because of the view that formal change control by high-level management over the design is not necessary or even desirable during the development stage. This does not mean that the design is concealed, as will be clear from the discussion of configuration control below.

Baselines are used for communication between management, the development team, and the end users of the software. They provide reference points for system transformation, auditing procedures and change control. This is especially important dur-

ing the maintenance stage when development of changes to a component may take place while the old version of the component is still being used in the running system.

2.2 Configuration identification

2.2.1 Configuration structure

During the development of the software system, design concepts are gradually refined and transformed into the building blocks from which the runtime system will be built. It is important that this process is reflected in the identification of the hierarchical configuration structure, in order to ensure that correspondence between components derived from one another is maintained.

For the purpose of this paper, a subsystem or computer program configuration item (CPCI) is defined as a functional part of the total system which can be developed, tested and accepted as a unit, once its functional environment exists. Therefore, subsystems can be regarded as independent entites, each consisting of several types of components, eg documents, module listings or files utilising operating system facilities.

During the life cycle of a project, more and more components are defined and added to the configuration structure. The source of each, that is the file containing basic human-readable statements, should be uniquely identified and individually put under configuration management. This ensures that no component can be changed without authorisation, and that the possibilility of component loss is minimised.

2.2.2 Component identification

It is important to ensure always that the internal or machine-readable image of a software component is the latest approved version and can be traced back to the latest external or human-readable image.

In an environment where total control over the changing and rebuilding of machine-readable components is not practicable, the gains of maintaining control by component naming conventions are not worth the effort. Elaborate identification schemes such as described in [1] and [2] also often cause resentment among technical personnel. From a configuration management point of view it is therefore easier to manage only human-readable files, and rebuild the system from these, during the establishment of baselines.

Naming conventions and standards ensuring that disk file names are allocated uniquely, identifying the project and subsystem, together with the policy that only the latest file version is a part of the software system, works much better in practice.

2.3 Configuration control

The need for configuration control developed from the need for control over changes to a software system. However, configuration control is a more general function which comprises control over system development as well. Configuration control therefore starts at the same time as the project.

2.3.1 The configuration control board

The configuration control board (CCB) is a body which has the following responsibilities:

- Define the project baselines
- Approve, monitor and control the transformation from specifications and design concepts to software components
- Approve, monitor and control changes to a software system.

These responsibilities are exercised with the aid of the configuration auditing function discussed below. The CCB usually bases its activities upon a quality assurance plan (QAP) existing for each software project or for the entire installation. The QAP may for instance be based upon IEEE standards [3] and spells out procedures, responsibilities and management control documentation to be generated during the project life cycle. It also describes the interactions needed with the end users of the software as well as formal review and configuration management procedures.

2.3.2 Baseline changes

Baseline definition was discussed in 2.1.2. Formal baselines correspond to stages in the software life cycle when components are frozen, protected against unauthorised change, and used as a basis for subsequent development. However, once a baseline has been formally approved, subsequent developments may force changes to it or to previous baselines.

If these changes are in the nature of refinements to approved concepts, product integrity is not affected, and changes to the relevant documents can be handled in the same way as final polishing.

Changes as a result of optimisation procedures may indeed affect product integrity if previously approved concepts are changed by the optimisation. In this case, as in the case of formal change requests, the changes should be implemented in a series of steps, so that the changes can be controlled and approved to reach an update of the previously approved baselines.

An updated baseline should not be confused with the next-level baseline, which is formally recognised by management when the next stage in the software life cycle commences. For instance, once the specification documentation has been formally approved, subsequent design problems may force an update of the specifications, and therefore of the allocated baseline as well. In this case the next level baseline will only be reached at the end of the development stage (the product baseline).

Because each baseline provides a visible step in the software life cycle, the documents related to each life cycle stage must always be an accurate reflection of the actual baseline.

2.3.3 Change control

Change control applies to all changes which become necessary after a component has been completed and formally archived. This may happen after a life cycle stage is completed.

For instance, when a subsystem is completed and an acceptance test has been passed successfully, it must be archived and protected against unauthorised changes. The approval of the subsystem also implies that general components such as function routines which may be used in subsequent development are frozen as well.

The Configuration Control Board (CCB) must approve all change requests and ensure that all components affected by the change are identified. If, for instance a module must be changed, the program design document and the subsystem specification document must be changed as well, if affected, in order to ensure that all documentation stays up to date.

2.4 Configuration Auditing

2.4.1 Baseline approval

The configuration auditing function is used whenever a baseline has to be approved. This implies that all components belonging to that baseline must be approved.

Configuration auditing never conflicts with other quality assurance techniques, as product integrity is the common aim. However, configuration auditing encompasses more than just checking whether a component conforms to standards.

Firstly, the software configuration must be verified to ensure that all components can be traced from components belonging to the previous baseline and that no components have been omitted. For instance, the specification documents for all the subsystems must be derived from the system definition document, taking into account the configuration specification (which divides the system into subsystems) and the hardware specification (if any) as well. During the process it must also be ascertained whether existing components not previously put under configuration management are included as part of the software system.

Secondly, the configuration must be validated to ensure that each component fulfils its specified function.

During the implementation of change requests, the auditing function must be exercised to ensure that the changes stay within the approved limits.

2.4.2 Auditing storage procedures

The quality assurance plan (QAP) usually describes the required backup and storage procedures. Auditing procedures must ensure that:

- All files are on backup storage media
- Backup storage media are readable
- The configuration structure identifies all configuration components under development
- The disk file identified in the configuration structure indeed contains the latest version of the component
- Multiple copies of a component do not exist, except in the case of backup procedure
- All components have been inspected by the quality control official before final acceptance
- If the previous baseline has been archived, it is protected against change
- The archived configuration is complete
- All archived configuration components are the latest version as used in the runtime system
- During the processing of a change request no unauthorised changes are made.

2.5 Configuration status accounting

2.5.1 Development Traceability

When a listing of any source file under configuration management is studied, its development history should be immediately apparent. This should describe what events have happened during the development of the component, as well as when these events occurred. When this is not done, fundamental changes may have been made to the file since a previous version was approved, and tracking of changes becomes impossible.

2.5.2 Configuration reports

During the life cycle of the project, a resumé of the software configuration and its state of development must be available to project and general management. This is necessary both for post-mortem analysis and for future project estimates.

The configuration report should show the breakdown of the project into independent subsystems. It should also reflect the hierarchical breakdown of software components, showing clearly which components are global to the whole project (eg general function routines) or to a specific subsystem (eg special function routines).

The configuration report should also show the progress on the implementation of changes, grouping all relevant information so that attempts to initiate a change affecting a component still undergoing another change, become immediately visible.

3.0 A Configuration Management Environment

A discussion of the application of the above configuration management principles in a specific installation, follows below.

3.1 Management structure

3.1.1 Organisation

Every project being developed is the responsibility of a project manager. He plans the project and allocates the personnel available. He also enters the names of all components on the configuration reports. These reports are formally distributed to the members of the configuration control board.

The CCB consists of representatives of the end users, the buying organisation and the developing organisation on a high management level. The CCB delegates some of its functions as discussed in 2.3.1 to a working committee on a lower management level, which includes the project manager. Some of the configuration auditing functions are delegated to quality control officials, who are also responsible for ensuring that the quality assurance procedures stipulated in the QAP are carried out.

3.1.2 Management control documents

The Configuration Control Board functions with the aid of management control documents. Such a document is generated whenever a scheme for which funds must be allocated, crops up. A copy of the document is signed and filed whenever the CCB or quality control officials approve baselines or development stages during the resultant development life cycle.

A job request is a management control document identifying the initial development of a software project or parts of a project. A change request identifies changes to an existing software system and all relevant components. The third type of management control document comprises all kinds of reports. With the aid of management control documents, all development progresses in an orderly way, because this system ensures that all software components to be developed, are identified and put under management control.

3.2 Configuration management information

3.2.1 Component identification

Every component put under configuration management, is explicitly referenced by name. Only the following human-readable files are put under configuration management:

- Document source files
- Module files in a high-level or assembly language
- Files utilising facilities for compiling, task building, etc.

It should always be possible to rebuild the software system from a backup medium, either before or after acceptance testing. Because of this, for each program (or task) files utilising specific installation and/or supplier facilities, eg. a linkage editor file, are included in the software system, which can then be used to compile every module and build a task image. With the aid of such files, the system can be rebuilt by personnel from outside the development team. The same applies to the rebuilding of project libraries.

Function or macro libraries which are to be used by more than one project are managed as a separate project, so that changes can be properly controlled.

In order to facilitate control over document generation, the document classification indicates:

- The project
- The subsystem or computer program configuration item (CPCI)
- A document serial number within the subsystem
- The development stage
- The document type or standard against which it must be measured.

As modules are usually managed as part of a subsystem, the first two characters of module names identify the subsystem or CPCI.

3.2.2 Component Versions

Version numbers are used to identify the current version of a component. Changes to any component imply that all occurrences of the component are changed, so that only one version is in use at any one time. When an older version is to be kept, the new version receives a new component name.

All components are created with an initial version of 1.0. Version numbers play a major role in configuration management and are always included in the configuration reports.

The first version digit is increased when:

- the component is ready for acceptance testing after initial development.
- the component undergoes a fundamental change after it has been archived.
- the component is changed for a new system version.

The second version digit is increased when the component undergoes a change after:

- formal distribution
- a management document has been approved for execution
- acceptance by quality control
- an unsuccessful acceptance test
- initiation of a change request in the case of minor changes.

No version decreases are accepted.

3.2.3. Component Statuses

Components are created with provisional status. The following milestones reached during the life cycle of a component are regarded as a change in its status:

- Formally distributed (documents only)

— Approved for execution (management documents only)
— Approved by quality control
— Released prior to acceptance testing (software components only)
— Accepted.

Whenever a component receives a new version, its status is lowered to provisional. Therefore, the series of status changes may be applied to a component several times during its life cycle.

It is not necessary to use all the possible statuses, but for a specific version the changes must be in sequence.

3.2.4 Configuration Management Pages

At the time a component is created, two pages are added with configuration management information. The first page contains the component heading, classification or name, author, date of creation, latest version and its distribution list. The second page contains the development history of the component. Each version line contains the version number, version date, and the name of the management control document used for development of the component. Comments may be added below a version line to describe the reasons for version increases.

Standard layouts for specific module file types to aid quality inspection, are also added when a module is created.

4.0 Configuration Management Procedures

4.1 Configuration reporting

4.1.1 Development Configuration Report (DCR)

The DCR contains a complete list of all components under configuration management which are being developed for a specific project.

They are grouped according to the job number used for planning and costing purposes. Within each job, entries are indented to show the system hierarchical structure. A group of components developed under the jurisdiction of a management control document follows directly below the entry for the control document.

The following information is entered for each component:

— Component version
— Component status
— Status date
— Percentage completion.

4.1.2 Archive Configuration Index (ACI)

The ACI is created at the time the first subsystem has been accepted and archived. The ACI is updated after a successful acceptance test, and/or the formal approval of a baseline, to contain entries for all complete and archived components. Component version numbers in the updated ACI form the basis for future change control.

4.1.3 Quality control lists

After the DCR of a project has been updated, a quality control list is printed, containing the names of all software component version numbers in the updated ACI form the basis for by the quality control official when components have been inspected.

When the component is turned down, the percentage completion is lowered and the component will be listed on the quality control list again when it reaches 100 % completion.

4.2 Control over developing software

4.2.1 Creation

Before any component can be created, its name must be included in the development configuration report (DCR). This allows the project manager to exercise control over the creation of components, and to ensure that the DCR is an accurate reflection of the hierarchical structure of the software system.

If the component is to be used exclusively by a specific subsystem, this subsystem must be identified in the DCR at the time of the first entry. Modules not identified in this way are regarded as global components, used by more than one subsystem. Changes to such components may have far-reaching effects, especially if the subsystems concerned are not tested and accepted together.

4.2.2 Updating

During the development phase, updating of components can be done without any restrictions. However, version and status changes are taken into account and entered in the development history page.

Once a component has been formally released for acceptance testing, the source file is protected from change. These files are stored on configuration management library tapes.

After a component has been accepted and archived, it is not available for changes except in the case of a change request. These files are stored on archive tapes.

4.2.3 Backup procedures

Files are copied to configuration management library tapes in the following instances:

— 100 % completion
— a version increase when the component is already 100 % complete
— formal distribution
— approval by quality control
— release prior to acceptance testing.

4.2.4 Development status changes

All status changes higher than provisional are formally entered by responsible personnel.

4.2.5 Document distribution

Management control documents are formally distributed to the members of the configuration control board, when a copy is printed for each person, including the name of the person and the copy number on each page.

Documents forming part of the software system, are formally distributed when necessary as an aid to formal reviews or structured walk-throughs.

Receipt vouchers are printed for each person, which contain the names of the documents which were printed for him. These receipt vouchers can then be signed and filed.

4.3 Acceptance of a subsystem

4.3.1 Release prior to acceptance testing

When a component is ready for acceptance testing, it is formally released by the project manager. After initial development, the version of a component is increased to 2.0 to indicate that the end of its development phase has been reached. This is entered in its development history page as well.

After a group of components has been released, configuration management library tapes are made, and the original source disk files are deleted, to prevent further changes. This process is continued until all the components have been released and the source files of the complete subsystem have been transferred to tape.

At this stage the subsystem, including all libraries, is rebuilt completely from the components on the tape. If the rebuilding is not successful, there may be components which were not put under configuration management and this can be rectified. Although compiling and rebuilding takes time, it is the only possible method of ensuring that the subsystem is complete, and that it contains no embedded machine-readable images for which the sources may disappear in future.

When modules have been compiled during the rebuilding process, a listing is printed for filing. Listings of other components such as files utilising operating system facilities, are also printed and filed at this stage. Document copies for filing were printed when they were distributed, but if distribution has not taken place, they are printed as well.

After rebuilding, the complete subsystem is loaded on to an archive tape, duplicated and used for the acceptance test. If data files or other files are needed for the test, they are added as well.

4.3.2 Acceptance Testing

Procedures regarding the safekeeping of archive tapes, as well as rules governing the conduct of acceptance tests, are under the control of the configuration management board. The result

of the test is entered in the management control document and signed.

If the acceptance test is successful, the tape used for the test becomes the archive tape for the accepted subsystem.

However, if the test was unsuccessful, all components to be changed are unloaded from the tape. Their versions are increased to indicate this. No new change request is generated, as the original job or change request is still valid, and can be amended to indicate the changes. Further processing of the changes and repeating the acceptance test take place as described below for change requests.

4.3.3 Archiving the subsystem

At the stage when the acceptance test has been successful, an archive tape containing the complete subsystem already exists, and no source files are left on the development disk.

The entries for the accepted components in the DCR are transferred to the ACI, so that the subsystem hierarchical structure remains unchanged. The entries for the archived subsystem are removed from the DCR to indicate that it is no longer under development. However, the entry for the job request is retained, to show successful completion.

4.4 Changes to accepted subsystems

4.4.1 Initiating change requests

Change requests are formally initiated and approved by the configuration control board. The change request management control document should at least include a list of all component source files affected by the change so that unauthorised changes to other parts of the subsystem can be avoided.

The relevant source files, together with all other machine-readable images needed for testing are unloaded from the archive tape under the control of a responsible person. Before unloading, the names of the components are entered in the DCR together with the change request, to ensure control over the development of the change. The version of each component is also increased to indicate the change, and a brief list of the changes is added as comments to the development history page of the component.

4.4.2 Developing components under change requests

Development and status increases take place in the same way as for the original development under the job request as described above.

The component will be listed in the ACI with an accepted status. A higher version of the component will be listed in the DCR with one of the development statuses as described above.

The definition of a change request takes the criteria for the eventual acceptance test of the change request into account, so that changes can be developed and tested in stages without impairing system integrity. The original acceptance test is repeated in part or altered to test the changes.

4.4.3 Release prior to acceptance testing

The release procedure is the same as described above, with the exception that an archive tape of the subsystem already exists. When the tape is updated as described above, a duplicate of the previous running system is of course kept as well.

Those parts of the subsystem affected by the changes must be rebuilt before the acceptance test. It will not be necessary to rebuild the complete system when the changes were restricted to a few well-defined parts of the subsystem. If however, a function routine or macro were changed, the complete system is rebuild as before.

4.4.4 Accepting and archiving change requests

The procedures for accepting and archiving components developed under change requests are the same as those described for job requests above.

When the ACI is updated, an entry for the component exists already, so the entry must be changed to reflect the correct version and date of acceptance.

Entries are deleted from the DCR as before. However, the entry for the change request document itself is retained, to show successful completion of the change.

References

[1] E.H. BERSOFF, V.D. HENDERSON and S.G. SIEGEL, Software Configuration Management — An investment in product integrity, (Prentice-Hall, Engelwood Cliffs, 1980).

[2] R. DUNN AND R. ULLMAN, Quality assurance for Computer Software, (McGraw-Hill, New York, 1982).

[3] IEEE STANDARD for Software Quality Assurance Plans. IEEE Standard 730-1981.

[4] CONFIGURATION MANAGEMENT practices for Systems, Equipment, Munitions and Computer programs, MIL-STD-483 (USAF) (1979).

New reading

Mathematical Foundations of Programming **by Frank S Beckman. Published by Addison-Wesley, Reading, Massachusetts, 1980. 443 pages, exercises, chapter references, index.**

THE phrase 'mathematical maturity' is used to describe a way of thinking that is required not only of computer scientists but also of programmers, not only of systems analysts, but also of managers of programming projects and thus also of dp managers. Very few people would like to attempt a definition of this desirable quality, but one may indicate a relatively painless way of acquiring it: Peruse the book under review.

In its breadth of coverage (with one omission noted below), in the topics dealt with, in the way all topics are related to actual or potential computing practice, the book contributes to the acquisition of the vitally important quality of mathematical maturity and also a quality that can only be termed algorithmic maturity. In other words, this book contributes to the appreciation of what can but also of what cannot be computed, and the various ways in which a computation may be characterised, realised, criticised, and analysed.

The author covers the area variously known as 'theory of computation' or 'meta-theory of computing', in a descriptive, almost intuitive, but mathematically sound fashion. The topics covered include: The concept 'effective'; functions and sets; recursive functions; computability and its limitations; automata and languages; and computational complexity.

In every chapter the author introduces the main topics by examples (so necessary for the development of the 'feel' for a subject) and then progresses through mathematical descriptions to applications. In the process many sidelines are touched upon which in textbooks are mentioned but then tantalisingly ignored. This is thus a book not only to be read but to be dipped into.

The practitioner in computing will find the book worth reading, not only to acquire the aforementioned maturity, but also to gain a background knowledge of the many developments in progress all over the world which go by the name 'fifth generation'. It is rather unfortunate, however, that the predicate logic receives but scant attention in the book since the Japanese fifth-generation R and D is based on Prolog-programming in logic. The research thrust in America, by the MCC, is to be based on a Lisp foundation and various aspects of this functional style are covered in the book.

For students taking a formal course at honours level in one or more topics covered by the author, the book should be recommended reading in order to put their field of study in perspective. For honours students not taking such topics, the book should be required reading.

STEF W POSTMA

Notes for Contributors

The purpose of this Journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review articles, exploratory articles of general interest to readers of the Journal. The preferred languages of the Journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to: Prof. G. Wiechers at:

Department of Computer Science
University of South Africa
P.O. Box 392
Pretoria 0001
South Africa

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. The original ribbon copy of the typed manuscript should be submitted. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name, and the affiliation and address. Each paper must be accompanied by a summary of less than 200 words which will be printed immediately below the title at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Illustrations and tables should not be included in the text, although the author should indicate the desired location of each in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Illustrations should also be supplied on separate sheets, and each should be clearly identified on the back in pencil with the Author's name and figure number. Original line drawings (not photoprints) should be submitted and should include all relevant details. Drawings, etc., should be submitted and should include all relevant details. Drawings, etc., should be about twice the final size required and lettering must be clear and "open" and sufficiently large to permit the necessary reduction of size in block-making.

Where photographs are submitted, glossy bromide prints are required. If words or numbers are to appear on a photograph, two prints should be sent, the lettering being clearly indicated on one print only. Computer programs or output should be given on clear original printouts and preferably not on lined paper so that they can be reproduced photographically.

Figure legends should be typed on a separate sheet and placed at the end of the manuscript.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters between the letter O and zero; between the letter l, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetical order of author's name, and cited in the text by number in square brackets. Journal references should be arranged thus:

1. ASHCROFT, E. and MANNA, Z. (1972). The Translation of 'GOTO' Programs to 'WHILE' Programs, in *Proceedings of IFIP Congress 71,* North-Holland, Amsterdam, 250-255.
2. BÖHM, C. and JACOPINI, G. (1966). Flow Diagrams, Turing Machines and Languages with only Two Formation Rules, *Comm. ACM,* **9**, 366-371.
3. GINSBURG, S. (1966). *Mathematical Theory of context-free Languages,* McGraw Hill, New York.

Proofs and reprints

Galley proofs will be sent to the author to ensure that the papers have been correctly set up in type and not for the addition of new material or amendment of texts. Excessive alterations may have to be disallowed or the cost charged against the author. Corrected galley proofs, together with the original typescript, must be returned to the editor within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Fifty reprints of each article will be supplied free of charge. Additional copies may be purchased on a reprint order form which will accompany the proofs.

Only original papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.

Hierdie notas is ook in Afrikaans verkrygbaar.

Quaestiones Informaticae

Contents/Inhoud

*Presented at the Third South African Computer Symposium held on 14th to 16th September, 1983.