The MultiMedia Maintenance Management (M^4) System

Rachel J. McCrindle

Applied Software Engineering Research Group, Department of Computer Science, The University of Reading, Whiteknights, PO Box 225, Reading, Berkshire, RG6 6AY, UK
Tel: +44 118 931 6536, Fax: +44 118 975 1994
r.j.mccrindle@reading.ac.uk

Abstract. Although adoption of a software process model or method can realise significant benefits, there is generally a need to provide a level of computerised support if it is to be usefully applied to large real-world systems. This is particularly true in relation to the software maintenance discipline, where many of the problems to date have typically arisen from deficiencies in recording and being able to easily access any knowledge regained about a system during maintenance. The MultiMedia Maintenance Management (M^4) system has been designed and prototyped as a meta-CASE framework in order to promote maximum process flexibility and product extensibility. As such, a variety of bespoke or host-resident tools for activities such as process management, information reclamation and documentation, configuration management, risk assessment etc. may be plugged into the M^4 system and activated through a universal front-end.

1 Introduction

The last decade has witnessed an explosion in terms of the abundance, size and complexity of software systems being developed such that they now play a key role in almost every aspect of today's society [11]. Indeed, the software industry may be seen as one of continual growth: in the industrial sector as automatic control of many systems, previously unrealisable due to hardware limitations, becomes possible through the transference of complex algorithms to a software base [18]; and in the business arena where software plays an increasingly central role in business process optimisation and redesign [7]. Commensurate with the increasing reliance being placed on software is the need for cost effective, high quality software products that meet customer expectations, perform reliably and safely [3] and which can evolve to meet the changing requirements of a dynamic industry [1]. Additionally, recognition of software as a key corporate asset is now becoming evident and the importance of maintaining this asset is gaining momentum [16].

The importance of investing in the maintenance and control of software is further substantiated by the fact that in many of today's computer systems the hardware expenditure can no longer be considered the major factor in any costing scenario [17]. The high cost associated with software may be attributed to a number of factors linked not only to the human-intensive nature of the process but also to the characteristics of the software itself. The enormity of the task is evident when we consider that even a decade ago software systems were being described as the "*most intricate and complex of men's handiworks requiring the best use of proven engineering management methods*" [5]. Since then, the internal complexity of software has progressively risen as has the number and heterogeneity of components.

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2074, pp. 459–469, 2001.

Other factors to contend with include the increased linkage of software with its environment, firmware and third party components, wider application bases, changes in architecture over the lifetime of a system, fragmentation of upgrade paths, the increasingly distributed nature of software products and the need to link them with supporting information [12].

This growing awareness of software has precipitated an increase in the research being conducted within the software process model arena and the co-requirement of developing automated tools to support the resultant models. This paper describes the M^4 system created to support the development, evolution and maintenance of large-scale software systems.

2 Automated Lifecycle Support

Various maintenance process models exist [1, 2, 4, 10, 12] which address the maintenance process from a number of different perspectives (e.g. economic, task-oriented, iterative, reuse, request driven, reverse engineering). Although these models describe the maintenance process in varying levels of detail, they all centre on the evolutionary nature of software. In addition other characteristics shown to be important for the production of high quality long-lasting software include the ability to enable effective communication, to support cost-effective maintenance, to facilitate a re-useable process, to support evolution by serving as a repository for modifications and to facilitate effective planning and increased understanding of the systems being maintained. As such the M^4 has been developed with a number of key features in mind:

Control: an underlying core set of facilities should be integrated into the toolset to satisfy the requirement to regain and keep control of a system in a defined and consistent manner.

Adaptability: the framework should enable the integration of different tools into the toolset to support the individual working practices and methods of different maintenance organisations.

Extensibility: information gathered from other sources or tools should be able to be brought into the framework and maintained as a coherent set of data.

Evolution: new technologies should be exploited as they come into the mainstream computing community.

3 Development of the M^4 System

The M^4 system has been developed primarily to provide semi-automated support for the ISCM (Inverse Software Configuration Management) process model [12] and its associated PISCES (Proforma Identification Scheme for Configurations of Existing Systems) method [13]. However, due to its development as a flexible and open meta-CASE framework rather than a rigidly defined more traditional CASE tool [9] it can be adapted to incorporate and support other process models if required. The components currently incorporated in the M^4 system are shown in Figure 1.

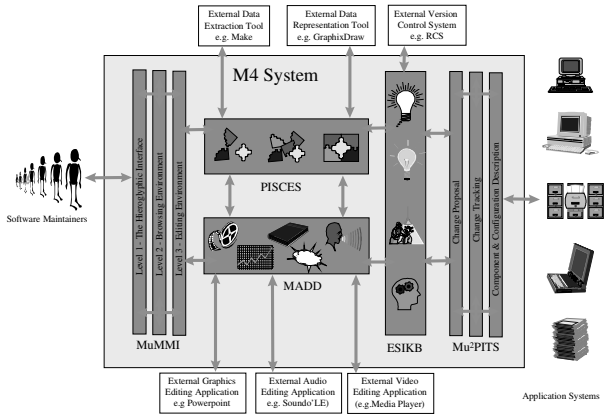

Fig. 1. Overview of M^4 System

3.1 EISKB

The Extensible System Information Knowledge Base (EISKB) forms the core of the tool. The purpose of the EISKB is to stored the rules, component details and domain knowledge pertaining to a system configuration. The ESIKB as it is currently implemented is actually an amalgamation of technologies spanning simple flat files, relational database systems and the inherent storage mechanisms of the host system. This rather piecemeal approach has arisen due to the development of individual tools which handle data storage locally but whose data can be shared through in-built data exchange mechanisms or via simple parsing tools which extract relevant data from the different toolsets as required. Whilst this approach has worked successfully and can exploit fully the advantages of flexibility there are also arguments for having a central M^4 repository, with strictly defined data structures, for use by all tools within the M^4 framework. However whilst this would offer more seamless tool integration and would lessen the likelihood of maintaining any redundant data within the M^4 system it may also make the incorporation of proprietary tools difficult or even impossible.

3.2 PISCES

The activities of the PISCES (Proforma Identification Scheme for Configurations of Existing Systems) tool are primarily concerned with the extraction and recording of information regarding the components and configuration of an application and the overall environment in which they play a part. As such the tool primarily implements a series of defined templates for incrementally building-up software system configuration information, although close linkages are also formed with the information recorded by the Mu^2PITS system and with any host system tools that aid the extraction of information about a system's components. The following facilities are offered by the PISCES tool [13].

Reclamation of configuration information: a series of templates are provided as an on-line guide to the consistent information collection and documentation of any software system being maintained. As maintenance occurs, information may be successively added and saved using one of two approaches: the templates may be updated and saved in the normal manner or a 'baseline' may be struck and the template saved as a new version. The point at which baselines are struck is definable

by the maintenance organisation, for example, this may be on a temporal basis or on a per maintenance basis after a change or series of changes have been made. Progressive baselining of the templates enables an evolutionary history of an entire system to be recorded.

Creation of HTML links and web pages: by creating the templates within the Microsoft Office suite of programs, the facility to add HTML links and covert the templates to web pages is enabled. Maintenance across the web then becomes a possibility especially if maintainers can link up via a web communications system [8].

Dependency information extraction and viewing: linkage to tools such as awk, grep and Make for extraction of dependency information; access to the resultant components such as Makefiles; and study of the code when combined with the maintainers expertise enables dependency information to be recovered and documented. Additionally small bespoke utilities enable features such as the resultant dependency tree structures to be displayed.

File location information and maps: the physical file locations for components can be identified and displayed through the use of simple data extraction utilities on either a per component or per system basis.

Incremental annotation facility: provision of a simple editor enables notes to be made regarding the understanding of the system gained during the comprehension process, or as a means of providing a temporary note to the maintainer, for example, to flag an activity that still needs to be carried out or to warn other maintainers of a troublesome section of code etc.

3.3 MADD

The MADD (Multimedia Application Documentation Domain) environment supports the management and control of, and access to the underlying multimedia types. A 'natural-language' control centre provides the primary linkage mechanism within the environment enabling links to be established between the different project documents. The key facilities provided by the MADD environment are [14]:

Viewing and loading of stored documentation: this facility enables the multimedia documentation previously stored on a project within the M^4 system to be viewed. This includes the ability of the M^4 system to display recorded video, to play audio, to display 'pure' text or texts with incorporated animation or graphics, and to display statistics in a graphical or animated format. The system also enables concurrent viewing of different multimedia attributes, for example, it allows code to be viewed alongside an associated video recording.

Creation and saving of the documentation: this facility enables new multimedia documentation to be input into the M^4 system, stored within the M^4 environment as a project directory and subsequently accessed through the MuMMI interface.

Editing of stored documentation: this facility enables changes to be made to stored documentation. This is required to keep the documentation up-to-date and concurrent with the state of the software project or simply to make corrections to erroneous documentation. In the case of audio and video files, these may be externally edited and replaced in the system, or the editing tools may be loaded and the file edited through the M^4 system itself. Text documentation may be displayed as 'pure' text or may be associated with other multimedia files. For example, whilst in edit mode the user can use the audio tool to record and associate a voice-over with a piece

of text, thereby adding value to and aiding the understanding of an otherwise less descriptive pure text file.

Appending to stored documentation: this facility enables new information to be appended to the existing versions of the documentation held within the M^4 system. Although closely allied to the editing function, implementation is handled differently as changes are sequential in nature thereby building up a change history of a particular component.

Deletion of documentation: this facility enables documentation to be deleted either as complete multimedia files or as linkages to files. However, this should be supported with the disciplined practice of archiving the documentation before allowing it to be deleted from the project environment.

Provision of security and access rights: this facility provides a security mechanism in the form of password access. This is important because only certain personnel within an organisation may have the rights to change project documentation.

Data storage and retrieval: this enables the multimedia documentation accessible by the M^4 system to be stored in the form of files in the user directory. There is no restriction on the organisation of these files, thereby allowing the user to adopt a preferred method of operation. This can involve the storage of independent files associated with each project in individual directories. Alternatively, storage can be via a link to a database application, as occurs in connection with the information stored within the Mu^2PITS tool. The advantage of the latter method is the provision of a more organised framework and the ability to conduct more complex queries and searches.

It is also acknowledged that a fully configured M^4 system will store large quantities of multimedia documentation, in the form of external files. The need for a high capacity storage medium is determined by the size of video and audio files. However this technical problem of storage is becoming less of an issue due to the development of efficient compression methods such as MPEG, increases in computation power even in low-end machines, and the continuing reduction in the price of hard-disk and removable storage.

3.4 Mu2PITS

The Mu^2PITS (MultiMedia Multi-Platform Identification and Tracking System) tool supports documentation of the identifying features of system components and their integration into software configurations. The nature of the tool is such that it enables attributes such as the relationships existing between different file types to be recorded and the location of these components to be tracked across distributed networks. In this respect Mu^2PITS differs from many of the traditional configuration management tools which tend to concentrate on text based products. Mu^2PITS also supports the production of change requests and tracking of the status of changes throughout the maintenance process. The key facilities (Figure 2) provided by the Mu^2PITS tool are [6].

Documentation of component attributes: this facility enables information about a particular component to be documented. Once created the resultant component records may be amended, versioned or deleted. The documentation supports all types of multimedia components as well as the more traditional text-based components.

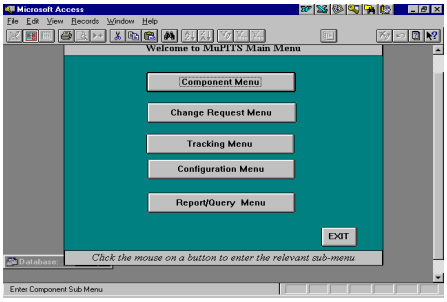

Fig. 2. Mu^{2}PITS main menu

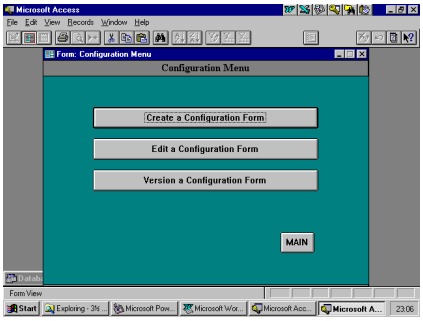

Fig. 3. Process configuration form menu

Documentation of configuration composition: this facility enables the documented components to be associated with one or more configurations. In this way master configuration lists of the components of configurations can be built-up. Additionally, dependencies on a per component basis can be recorded. Once created the resultant configuration records may be amended, versioned or deleted (Figure 3).

Creation/amendment/deletion of a change request: this facility enables a change request to be generated for a particular component. In order to avoid possible conflicts and loss of valuable data the system only allows one active request per component at any one time. The resultant change request form may be updated if the change request has to be amended in some way after scrutiny by the change control board and is archived once the approved change has been completed or immediately if the change has been rejected.

Creation/amendment/deletion of a status tracking form: this facility enables the status of an accepted change request to be tracked during the maintenance process and it thereby allows the status of the change itself to be monitored. A status tracking form cannot be raised until the change request form has been completed and agreed. The status attribute of the tracking form is amended during the change process to reflect the status of the component and the form is archived once the change has been completed.

Creation of reports and queries: this facility enables information to be obtained regarding the data held within the Mu^{2}PITS database. Information may be output to a printer as reports or to the screen as queries. The reports range from listing of component details to master configuration lists to management statistics, for example details of how many components are undergoing changes at a particular point in time.

3.5 MuMMI

The MuMMI (MultiMedia Maintenance Interface) is the front-end to the M^4 system, and is used to display the multimedia documentation [14]. This documentation is any information of relevance to understanding the high-level design or low-level code of a software product. It is routinely added to the system throughout maintenance thereby keeping the documentation up to date in relation to the underlying software product. The MuMMI co-operates closely with the MADD and is based on three levels of access:

Level 1 - MuMMI Manager: this level is used to select the particular project or system undergoing comprehension (Figure 4). Selection is via a graphical menu of multimedia attributes and determines whether it is the graphical, textual, audio or video documentation associated with a project that is initially displayed. Once a valid project file has been selected control is automatically switched to level-2 of the interface.

Level 2 - Browsing MuMMI Project Environment: this level is organised with respect to the selected project. Initially only the selected multimedia view of the project documentation and the control centre are displayed. The control centre uses a 'natural language' command interface in order to fulfil the requirements for fast operation by an experienced user group. The user communicates with the system through the entry of defined commands. A history of the commands entered within a session is maintained and can be invoked again by simple selection. As the program comprehension process proceeds the user can enter commands to display other multimedia views of the project related to the initially opened file.

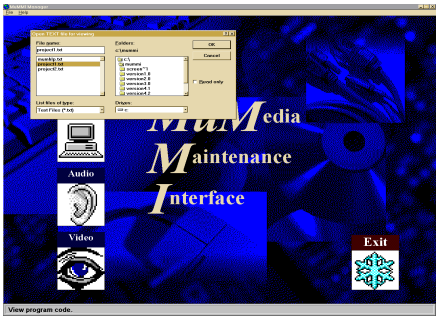

Fig. 4. Project selection window

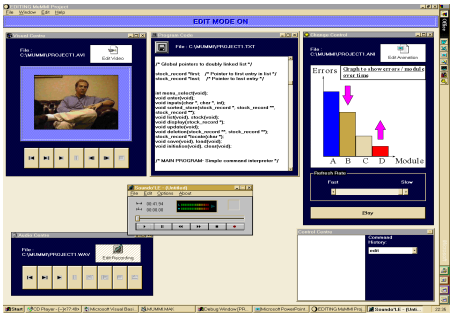

Fig. 5. Display of multimedia files

The interaction between this level of the MuMMI and the MADD component of the M^4 system is one of read-only access. Thus the user is able to browse and display the multimedia documentation on accessible project files but is prevented from making any changes to stored material. The user can enter the editing levels of the M^4 system through a command in the control centre followed by access via a password protection mechanism.

Level 3 - Editing MuMMI Project Environment: the interaction between the third level of the interface and the MADD component and is that of read- and write-access. Thus at this level the user is able to edit the existing multimedia documentation and to create and store new documentation within the M^4 framework. This facility allows update or extension of the information stored on an existing product as it is recovered during the program comprehension process (Figure 5).

3.6 M^4

The Multimedia Maintenance Manager (M^4) provides the overall framework of the system, binding together the ESIKB information storage mechanism, the MuMMI front-end and the MADD back-end. It also provides hooks into the bespoke Mu^2PITS and PISCES tools as well as enabling links to be made to external point function tools such as those for version management and information extraction. The evolution of

the M^4 system has also incorporated animated models of the maintenance process (Figures 6 & 7).

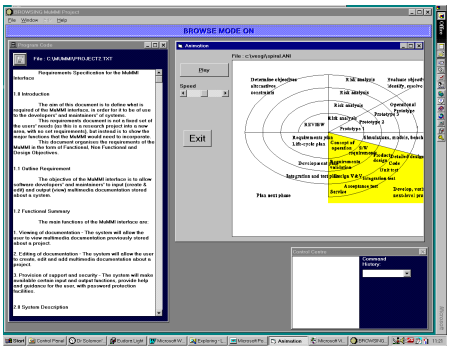

Fig. 6. Animated view of the spiral process procedures & personnel

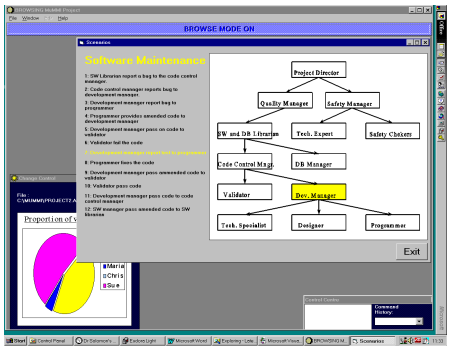

Fig. 7. Animated company maintenance

Additionally with such a dynamic environment as the M^4 system and the emphasis that needs to be placed on keeping documentation concurrent with the state of the software, there is a need for rigorous change and version control. Software configuration management requirements must also extend to the ability of the M^4 system to support a family of concurrently released software product configurations to different clients. This latter requirement of recording component and product configuration details has been addressed in the Mu^{2}PITS tool. However Mu^{2}PITS does not deal with the issue of change to and subsequent storage of the actual components themselves. As the M^4 documentation is in different multimedia formats the problems of managing changes are more complex than for text management alone particularly if the changes are to be stored as a series of deltas rather than complete files. Currently, links have been provided to an external, but tailorable change and version control system for multi-platform code management and work is underway on a fully-distributed multimedia configuration management system [15].

4 M^4 System in Use

Section 3 has described and discussed the functionality offered by each of the main components of the M^4 system. It is also important to establish the use of the M^4 system within the task-oriented context of the way in which maintainers perform their role. It is intended that the M^4 system is used during the program comprehension process prior to each maintenance change according to the following procedures:

Browse: initially, the maintainer browses the program code and any associated documentation already present in the MuMMI system that appears to be of use to understanding the proposed change and the affected parts of the system.

Evoke: if appropriate, the maintainer evokes any data extraction tools etc. existing within or integrated into the M^4 framework in order to generate additional understanding or dependency information about the construction of the system.

Complete: the maintainer documents any additional findings about the affected parts of the system within the specific template generated for the system under study.

Update: as the comprehension process occurs the maintainer gains more information about the application itself, the application domain, the detailed functionality or design of the system etc. This understanding is incrementally recorded using an appropriate multimedia type and subsequently stored within the M^4 system.

In essence the M^4 system acts as a 'maintenance oracle' capturing all previous maintenance experience within a single environment. The effects of this system are three-fold:

Increased understanding: as domain, system and application knowledge is incrementally built-up and recorded so too is the understanding of the system and its importance in the context of the organisational and system domain better understood. This has the effect of ensuring that changes can be prioritised towards those of a key business nature and it also ensures that changes can be made more safely with reduced risk of the ripple effect occurring.

Faster changes: as more information is built-up about a system the time attributed to the comprehension process should become successively reduced. This is because the prior knowledge of the maintainer themselves and of any previous maintainer is available for use and hence provides a 'head start' so that the time taken to understand a change and its impact on the rest of the system can become progressively more rapid.

Foundation for re-engineering: the increased documentation and understanding of a system, as well as notes about a system or collection of metrics about errors etc. assists with identifying the areas of the system that will most benefit from being re-engineered. Additionally, identification of the components and records of how they are synthesised into complete configurations provides a solid foundation on which the actual re-engineering process can be based.

Additionally, although the use of the M^4 system has primarily being discussed within the context of its use during maintenance, maximum benefit of the M^4 system comes as a result of it being used from the inception of a green-field project. In these circumstances, the M^4 system can be populated with the complete set of components and documentation about an application. Additionally, all design decisions, key meetings or reasoning of an informal, semi-formal and formal nature can be captured from the outset of development through a combination of the different multimedia attributes. Used in this way, the M^4 system pre-empts the maintenance process and acts as a 'maintenance escort' during hand-over of the application [14].

5 Conclusions and Benefits

The M^4 system framework may be summarised as having a multimedia (MuMMI) front-end, an underlying repository (ESIKB) and a control and management back-end (MADD). In addition to this there is a data collection and collation mechanism (PISCES), support for the SCM activities of identification, control, status accounting and audit (Mu^2PITS), and the integration of other tools such as those for data extraction and version control. These tools plus their ability to link to other tools may be considered to provide a complete maintenance environment. In summary the benefits afforded by the M^4 system are:

Role: the M^4 system supports the ISCM process and PISCES method but can be readily adapted to incorporate other process models.

Flexible: adoption of a meta-CASE approach to the development of the M^4 system enables it to be very flexible in terms of its functionality, and readily extensible with regard to the range of tools that it can incorporate.

Generic: as well as being adaptable for different operational platforms, the system may be tailored to suit different application domains and programming languages through the on-line implementation of the PISCES templates.

Cost effective: many current maintenance tools are costly to buy and implement. The M^4 system can be integrated with tools already residing on the host system thereby reducing cost, and minimising the disruption associated with training maintainers to use new tools.

Easy to use: as the PISCES method provides guidance rather than prescription for the maintenance process, the different working practices of maintainers can be accommodated within the framework of the system whilst still providing the required consistency and control.

Multimedia: the M^4 system makes use of a hypermedia approach to enrich the maintenance and evolution of software systems. It also provides the MuMMI interface as a means of managing the recording, activation and dissemination of multimedia material pertaining to software system configurations.

Domain Knowledge: The M^4 system pays particular attention to being able to capture domain knowledge about a system undergoing maintenance. This is facilitated by exploitation of the multimedia capabilities described above.

Communicative: maintainers have expressed the need for better communication between the development and maintenance teams. This includes the need for the initial transfer of knowledge between the two teams to be as complete as possible, as well as the requirement for long term conservation, dissemination and refinement of expertise from one maintainer to another. The long term transfer facility is provided by using the M^4 system during maintenance whilst initial transfer of knowledge is facilitated if the M^4 system is used from the outset of the development process.

Transferability: a by-product of the M^4 system development is the transferability of the underlying framework into other realms of information management. Although the M^4 system has centred around providing an environment for the development and maintenance of software systems it has become evident that there is enormous potential for expansion of the ideas into many other application areas requiring production and control of a mixed media type.

Although a working prototype exhibiting the above characteristics has evolved during the course of the research, there are still a number of ways in which the M^4 system can be improved both in relation to the degree of functionality offered by the M^4 system and in relation to the quality of development of the prototype system. For example, work is continuing on various aspects of the tool associated with risk management, project documentation frameworks, enhanced web and multimedia support. The M^4 system was developed as a proof of concept meta-CASE system and features of the tool have attracted considerable interest. For example a specific version of the tool concentrating on enabling large-scale knowledge management and personnel communication both locally and across the Internet has been developed for a major international company.

Acknowledgements

Thanks must go to Stuart Doggett and Kirsty-Anne Dempsy, final year project students and Frode Sandnes, Research Assistant, on the VES-GI project for their help in implementing parts of the tool-set. Much appreciation is also due to Professor Malcolm Munro for his support in formulating the ISCM process model and PISCES method.

References

1. Basili, V.R., *Viewing Software Maintenance as Re-use Oriented Software Development*, IEEE Software, Vol. 7, pp19-25, Jan. 1990.
2. Bennett, K.H., Cornelius, B., Munro, M. and Robson, D., *Software Maintenance*, in J. McDermid, ed. Software Engineer's Reference Book, Chapter 20, Butterworth-Heinemann, 1991.
3. Bhansali, P.V., *Universal Safety Standard is Infeasible - for Now*, IEEE Software, pp. 8-10, March 1996.
4. Boehm, B.W., *Software Engineering*, IEEE Transactions Computers, pp. 1226-1241, Dec. 1976.
5. Brooks, F.P., *The Mythical Man Month: Essays on Software Engineering*, Reading, Mass., Addison-Wesley, 1982
6. Dempsey, K-A., McCrindle, R.J. and Williams, S., *Multimedia Multi-Platform, Identification and Tracking System (Mu^2PITS)*, Final Project Report, Supervised by R.J. McCrindle, Department of Computer Science, the University of Reading, 1996.
7. Georges, M., Message from the General Chair, Proceedings, *International Conference on Software Maintenance*, France, 1995, IEEE Computer Society Press, 1995.
8. Hill, S. and McCrindle, R.J., *The Virtual Body Project*, Draft Paper, March 2001.
9. Iivari, J, *Why are CASE Tools not Used?*, Communications of the ACM, Vol. 39, No.10, pp. 94-103, October 1996.
10. Lano, K. and Malic, N., *Reengineering Legacy Applications using Design Patterns*, In Proceedings, Eighth International Workshop on Software Technology and Engineering Practice, pp. 326-338, London, July 1997
11. Lehman, M.M., Software'e Future: Managing Evolution, *IEEE Software*, pp. 40-44, January-February, 1998.
12. McCrindle, R.J., *Inverse Software Configuration Management*, PhD Thesis, The University of Durham, 1998
13. McCrindle, R.J. (nee Kenning) and Munro, M., *PISCES - An Inverse Configuration Management System*, Chapter 17 in Reuse and Reverse Engineering In Practice, Ed. P.A.V. Hall, Chapman & Hall, 1992
14. McCrindle, R.J., and Doggett, S. *The Multimedia Maintenance Interface System*, in Proceedings COMPSAC 2000, Taipei, Taiwan.
15. O'Connell, P. and McCrindle R.J., Using SOAP to Clean Up Configuration Management, Draft Paper, March 2001.
16. Parnas, D.L., *Software Ageing*, In Proceedings 16th International Conference on Software Engineering, pp. 279-287, 1994.
17. Shapiro, S., *Splitting the Difference: the Historical Necessity of Synthesis in Software Engineering*, IEEE Annals of the History of Computing, Vol. 19, No. 1, pp. 20-54, 1997.
18. Warwick, K., *An Introduction to Control Systems*, 2nd Edition, Advanced Series in Electrical and Computer Engineering, Vol. 8, World Scientific, 1996.