

/A SYSTEM OF AUTOMATED TOOLS
TO SUPPORT CONTROL OF SOFTWARE DEVELOPMENT
THROUGH SOFTWARE CONFIGURATION MANAGEMENT/

by

MARTHA GEIGER WALSH

B. S., Muhlenberg College, 1969

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree


MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985

Approved by:


Major Professor

LD
2668
.84
1985
W347
C. 2

ACKNOWLEDGMENTS

111202 964926

I would like to thank several people who made my sojourn at Kansas State University both rewarding and enjoyable: Dr. William J. Hankley and Dr. Virgil E. Wallentine for their review of my Master's Report and their comments; Mick and Mary Beth Cole for their help and support in all areas of the AT&T program; and especially Dr. David A. Gustafson and Mrs. Karen Gustafson. I want to express my appreciation to Dr. Gustafson for his encouragement, pertinent suggestions and generous giving of time, even while on vacation, which was certainly above and beyond the call of duty; and to Mrs. Gustafson for supporting her husband during the demands of our project and for always being gracious to us.

TABLE OF CONTENTS

CHAPTER 1. OVERVIEW.....	1
1.1 Introduction.....	1
1.2 Literature Review.....	2
1.2.1 Software Development and the Fifth Generation.....	2
1.2.2 Software Development and Software Configuration Management.....	3
1.2.3 Software Development and Control.....	8
1.3 The SCM Control Project in Relation to the Literature.....	17
1.4 Summary.....	19
CHAPTER 2. REQUIREMENTS OF THE SCM CONTROL SYSTEM.....	21
2.1 Scope of Implementation.....	21
2.2 General Requirements.....	21
2.3 Specific Requirements.....	23
2.4 Functional Requirements.....	23
2.5 Design Decisions Based on the Requirements.....	25
CHAPTER 3. DESIGN OF THE SCM CONTROL SYSTEM.....	26
3.1 Overview.....	26
3.2 Primary Shell Specifications.....	32
3.3 Main Programs.....	32
3.3.1 Submit a Modification Request.....	32
3.3.2 Update a Modification Request.....	33
3.3.3 Select Modification Requests.....	35
3.3.4 List a Modification Request.....	35
3.3.5 Validate Modification Requests.....	36
3.4 Subprograms.....	37
3.4.1 Check Modification Requests.....	37
3.4.2 Authorize Update of a Configuration Item.....	37
3.4.3 Get a Modification Request Record.....	38
3.4.4 Setup Path of CI Files.....	39
3.4.5 Call an SCCS Function.....	40
3.4.6 Open a File in the MR System.....	40
3.4.7 Close a File in the MR System.....	41
3.4.8 Setup Path of MR Administrator.....	41

3.5 MR System Directory and File Structure.....	42
3.6 Modification Request Record.....	44
CHAPTER 4. IMPLEMENTATION OF THE SCM CONTROL SYSTEM.....	47
CHAPTER 5. CONCLUSIONS.....	51
5.1 Evaluation of the Implementation.....	51
5.2 Possible Extensions.....	52
REFERENCES.....	54
APPENDIX A. MODIFICATION REQUEST SYSTEM INSTALLATION GUIDE.....	58
APPENDIX B. MODIFICATION REQUEST SYSTEM USERS' GUIDE.....	65
APPENDIX C. CONFIGURATION ITEM/BASELINE IDENTIFICATION SYSTEM INTERFACE.....	74
APPENDIX D. SOURCE CODE.....	79

LIST OF FIGURES

Figure 1.	Relationship of SCM to KSU Prototype Tools.....	22
Figure 2.	Data Flow Diagram of MR System (Submit and Update).....	29
Figure 3.	Data Flow Diagram of MR System (Select, List and Validate).....	30
Figure 4.	Hierarchy Diagram of Modification Request System.....	31
Figure 5.	Directory/File Structure of Modification Request System.....	43
Figure 6a.	Contents of a Modification Request Record.....	45
Figure 6b.	Contents of a Modification Request Record (cont.).....	46
Figure 7.	Directory Structure for MR System Installation.....	60
Figure 8.	Code Directory Structure for MR System Installation.....	61
Figure 9.	MR System Primary Function Menu.....	66
Figure 10a.	Keywords Used to Identify MR Fields to be Updated.....	69
Figure 10b.	Keywords Used to Identify MR Fields to be Updated (cont.).....	70
Figure 11.	MR Status Types Available for Selection.....	71
Figure 12.	Codes Used For Listing Modification Requests.....	72
Figure 13.	Project Oriented Structure for MR System/CI System Interface.....	77
Figure 14.	SCM Oriented Structure for MR System/CI System Interface.....	78

CHAPTER 1. OVERVIEW

1.1 Introduction

For most software development efforts to be considered successful, certain requirements must be met. These product and project criteria include user satisfaction with product integrity and management objectives to produce the product on time and within budget. In order to meet these requirements, software must be developed in a controlled environment. The discipline called Software Configuration Management can be applied to support controlled development of software.

This report documents a project whose objective was to define, design, implement and document a system of automated tools that would support the control of software development through the application of certain principles of Software Configuration Management, specifically Configuration Control and Configuration Status Accounting. This system is part of a larger research project being conducted at Kansas State University. The objective of the larger project is to develop a prototype of a software development environment that is adaptable for use with a group of computer systems called Fifth Generation systems. The implementation addressed in this paper can provide a method of controlling and tracking the components of the prototype as they are developed.

In order to provide some continuity between the overall project and the system that is the subject of this report, this chapter provides references to some areas of the Fifth Generation projects that are

applicable to software development environments. The chapter continues with a discussion of Software Configuration Management as it relates to software development. Following that topic, the subject of this project, control of software development, is described.

The remainder of the report describes the development of the control system. Chapter 2 identifies the requirements for the system and is followed by a description of its design in Chapter 3. The implementation and testing of the software are documented in Chapter 4. Lastly, Chapter 5 presents some conclusions and possible extensions to the project.

1.2 Literature Review

1.2.1 Software Development and the Fifth Generation

The phrase "Fifth Generation computer systems" generally refers to a group of projects currently under development which are intended to revolutionize the computer industry. One of the best known projects is sponsored by the Japanese government. The aim of the Japanese is to develop by 1990 a prototype system combining the concepts of knowledge-based systems, very-high-level programming languages, decentralized computing, and very large scale integration (VLSI) technology [Tr82].

Emphasis is placed in the first phase of the Fifth Generation project on the development of sets of tools to be used in the later phases [Ki83]. Supporting these objectives are research and development in the areas of development support systems and intelligent programming

systems [Fe83]. The "intelligence" of the software begins in these tools through techniques which enable man to interact with machine at the specification level. The tools allow system requirements to be stated in machine-understandable languages. The requirements can then be reviewed, mechanically tested for consistency and completeness, and used for prototyping and code generation [Ro84]. Therefore, the tools will aid in eventual production of software.

1.2.2 Software Development and Software Configuration Management

All development projects, especially those for Fifth Generation-related systems, have the potential for "missing the target." As complexity of the project increases, so do the risks of having the project fail. Projects conducted by both private industry and the Federal Government have been considered failures for various reasons, including not meeting the requirements of users and management.

What is needed is a way of avoiding such mishaps. The development effort must produce a system that both fulfills user objectives for the product and satisfies management objectives for the project. Unfortunately, this target is often a moving one. Change, especially as applied to software development, is inevitable so we must be prepared for it [Br79]. It is necessary to control and manage change rather than simply to react to change.

One method of controlling the system development effort and accompanying changes is to employ the principles of Configuration Management. As defined by the Institute of Electrical and Electronics

Engineers (IEEE), Configuration Management is "the process of identifying and defining the configuration items in a system, controlling the release and change of these items throughout the system life cycle, recording and reporting the status of configuration items and change requests, and verifying the completeness and correctness of configuration items" [IE83b]. A configuration is an arrangement of parts that bear some relationship to one another [Be80]. These parts or configuration items are elements that satisfy some function and are controlled by Configuration Management [IE83b]. A configuration item can be any component produced during the development of a system, from its initial concept to its phase-out, and includes hardware and software as well as documentation. For example, a configuration item could be a requirements specification document, source code, or a prototype computer.

Configuration Management has historically been applied to hardware and other physical configuration items of a system. The Department of Defense was a pioneer in this field, establishing broad policy in DoD Directives and documenting practices and procedures in Military Standards (MIL-STD). For example, DoD Directive 5010.19, entitled "Configuration Management," is a basis for many other Directives and Configuration Management efforts. A different type of document, MIL-STD-483 "Configuration Management Practices For Systems, Equipment, Munitions, and Computer Programs," provides specific forms and procedures to be used in Configuration Management [Br80].

However, software is becoming more dynamic than hardware. Due to the increasing complexity of software, the costs for its development have been increasing over hardware costs [Mc80]. In addition, it is becoming as necessary to control the functional configuration as it is to control the physical configuration developed to meet the system requirements [De80]. In the early phases of development, the system configuration consists of functional configuration items, such as performance requirements, output requirements and input specifications. The physical configuration, consisting of such items as computer terminals, graphical screens, program listings and user manuals, will be designed and implemented based on those functional requirements and specifications. Therefore, the control of the functional configuration is of growing concern.

These changing factors have led to the emergence of Software Configuration Management (SCM), which is "Configuration Management tailored to systems, or portions of systems, that are composed predominantly of software". Formally, Bersoff, Henderson and Siegel have stated that Software Configuration Management is "the discipline of identifying the configuration of a system at discrete points in time for purposes of systematically controlling changes to this configuration and maintaining the integrity and traceability of this configuration throughout the system life cycle" [Be80].

As stated in the definition of SCM, product integrity is a major objective in the development of systems. Product integrity can be defined as

"the intrinsic attributes

- which characterize a product that meets user requirements imposed or assumed or presumed or intended during any stage in its life cycle,
- which facilitate traceability from product conception (as an idea) through all subsequent stages in its life cycle, and
- which characterize a product that meets specified performance criteria.

In addition, the integrity of a product is diminished if the cost or delivery expectations of the buyer and user have not been fulfilled" [Be80].

This rather complicated definition can be summarized by stating that a product has integrity if it works as required by the user, if it can meet evolving user needs, if it is well designed and documented, and if it is delivered on time and within budget.

In order to attain and maintain product integrity, discipline must be applied to the development effort. To expand this idea, it might be said that "doing" disciplines and "support" disciplines contribute to the integrity of the software product [Be80]. By "doing" disciplines is meant the activities which collectively form the execution process or phases of the software development life cycle (for example, analysis, definition, design, implementation, testing).

The "support" disciplines constitute the planning, organizing, controlling and evaluation activities of the development effort. Often called "product assurance" disciplines, the support disciplines are comprised of Configuration Management, Quality Assurance, Verification and Validation, and Test and Evaluation. Quality Assurance is a collection of techniques and tools which are used to ensure that a

product meets, at minimum, a set of standards specified before the development of the product. Verification techniques are the methods used to ensure that a product meets predetermined objectives or specifications. Validation techniques are the methods used to determine that the product satisfies user needs and is consistent with the overall goals of the project. Test and Evaluation methods are used to exercise the product in order to assess its performance relative to the attainment of its objectives [Be80, Br82].

These product assurance disciplines are interrelated and, to a certain extent, overlap. All are concerned with controlling the product and ensuring that it meets specific objectives. These common concerns are often examined and made visible through the process of auditing the software development effort and configuration. This auditing function comprises the intersection of Software Configuration Management tasks with the other product assurance disciplines.

Four processes or tasks are involved in the overall performance of Software Configuration Management. These four processes are Configuration Auditing, Configuration Identification, Configuration Status Accounting, and Configuration Control. Configuration Auditing is the verification that each configuration item is consistent with and traceable to its predecessor and successor items, and the validation that each designated configuration meets the objectives stated in the requirements specifications of the system. Configuration Identification is the identification and description of the items, both individually and collectively, that comprise the configurations of the system at any

point in its life cycle. Configuration Status Accounting is the recording and reporting of significant events which affect the configurations, as well as the provision of information to support the other Software Configuration Management processes. Configuration Control is the control of changes and access to the configuration items throughout the system life cycle [Br80].

1.2.3 Software Development and Control

A software development project involves many complex and interrelated activities identified in the "doing" and "support" disciplines. Therefore, in order to meet project objectives, control over the system development cycle is not only desirable but also necessary. In fact, the very nature of the word "discipline" implies control. Control is accomplished through the interaction of people, tools, procedures and documentation. Each of the SCM tasks contributes to the control of the system development effort through one or more of these elements.

Configuration Identification aids in control through the naming, describing, documenting and numbering of configuration items. In addition, when items are related in a formal manner through releases, another level of control is introduced. Releases are also referred to as "baselines." A baseline is a collection of related configuration items that are formally described and fixed in status at a specific time during the system development life cycle [IE83b]. Often baselines are established for configuration items that exist at the end of a phase of

the life cycle. For example, a "functional baseline" would be established after the first phase of development and would include definition of the product and system requirements. An "allocated baseline" would include the division of functions between hardware and software [Be80, Du82]. Baselines aid in making configuration items visible and are really the beginning of their control.

Configuration Auditing aids in control of the configuration by requiring that changes to a preceding baseline be approved before inclusion in a new baseline. Because a baseline represents formal approval of the included configuration items, any subsequent baselines must be consistent in fulfilling the system requirements.

The Configuration Status Accounting task is responsible for recording and reporting of events pertinent to the development effort. These activities make the system development project and products visible, and therefore easier to manage and control. In fact, this task ties together all SCM functions by collecting, synthesizing and documenting data from all events. The resultant reports aid in control of the system development effort through communication of information, especially to project management.

At the heart of system development control are the mechanisms of Configuration Control. This SCM task is more than change control as in the generally accepted sense. However, the products of the development effort are constantly changing or evolving throughout the life of the system. Therefore, Configuration Control could loosely be considered

change control, just as Configuration Management is sometimes called the management of change [Du82].

1.2.3.1 Software Configuration Management Plan

Although it is not only a Configuration Control mechanism, the Software Configuration Management Plan (SCM Plan) nevertheless contributes to control of the software development effort. As Brooks has written, we must plan for change [Br79]. The SCM Plan is a tool for managing software development and, specifically, for planning for and controlling change during development. A Software Configuration Management Plan is a document which describes how the four tasks of Software Configuration Management are to be accomplished for a particular project. That is, it "documents the methods to be used for identifying software product items, controlling and implementing changes, and recording and reporting change implementation status" [IE83a].

The SCM Plan must be prepared at the start of the system development effort, before any of the "doing" discipline activities have begun. The Plan must describe how the activities of the development effort will be controlled. It should include a description of the organizations involved in the development effort and specifically the organization performing the SCM functions. The responsibilities and functions of the Configuration Control Board would also be included in the organization section. Another section of the Plan would describe the tools and procedures to be used to accomplish the SCM tasks. For

example, a description of the required baselines, or project milestones, would be documented, as well as the requirements for a Modification Request System. In addition, as in any management plan, the resources necessary to perform SCM must be identified in a section of the SCM Plan. These resources include costs, personnel required and equipment [Be80, IE83a].

The Software Configuration Management Plan is part of a set of overall project plans. The SCM Plan and its relationship to other project plans such as a Software Quality Assurance Plan is documented by the IEEE. This organization has prepared a standard describing the required and suggested contents of a Software Configuration Management Plan. A working group of this organization is also currently preparing a Guide to aid in the writing of SCM Plans [IE81, IE83a, Sc85].

1.2.3.2 Configuration Control Board

One of the most influential mechanisms of Configuration Control is the Configuration (or Change) Control Board (CCB). When configuration items have been reviewed and baselined, they are put under the supervision of a CCB. The CCB will evaluate subsequent proposed changes to those items and recommend disposition of the proposals. This evaluation includes feasibility, consistency, impact on quality and reliability, classification, priority, and cost and scheduling impact [Be84b]. The CCB represents the first step of the auditing process, which controls both the evolution of the system during the development life cycle and the revolution or changes to the system.

A Configuration Control Board is usually composed of representatives of the users, the developers and project management. As a group, the members represent the buyers and sellers of the system. In large organizations and for large projects, there may be a controlling CCB and lower level CCBs responsible for supervision of hardware and software as separate entities.

If Configuration Control is the heart of the SCM process, then the CCB is the heart of Configuration Control. Bersoff, Henderson and Siegel [Be80] have described overall Configuration Control and specifically the CCB as the "engine" of Software Configuration Management. In other words, it makes things happen. It drives activities. It does not grind them to a halt [Be80].

1.2.3.3 SCB Databases and Development Libraries

Another Configuration Control mechanism is a formal, standardized facility for storage of SCM-related information. Included in this mechanism are the concepts of the software development library and the database. A database can be used in the Configuration Status Accounting functions of collection and reporting of information such as configuration items, change requests, baseline contents and project status and costs. Once this information is captured, access can be controlled. Various database organizations and data models could be appropriately applied, depending on the individual application. One such scheme is an entity-relationship-attribute model which, according to Huff, can apply to such things as configuration items as well as

descriptions of control procedures [Hu81]. An entity-relationship-attribute model is a model which contains information about things (entities), associations between the things (relationships), and mappings (attributes) between the things and sets of values that describe or qualify the things [Ch76].

The SCM database is also described as an integrating and unifying medium for the software development environment. As such, specific tools such as a data dictionary can be used to inquire against and update the database without direct interface (coupling) or hindrance to other tools [Ho82]. The database tools and controls can be tailored to individual project requirements. As Sibley states, a Software Configuration Management Database Management System can be a "passive" or an "active" system. A passive system stores data as in the Configuration Status Accounting task. An active system additionally enforces controls such as requiring online entry of appropriate approval before a baselined configuration item can be updated [Si81].

A software development library or project directory is another repository of SCM data, most frequently source and/or executable code. As with the database, access and update privileges allow control of the information stored in a library or directory. The Source Code Control System (SCCS) supported by the UNIXTM Operating System is one example of such a storage facility [Su83]. However, this system is perhaps inaccurately named, because it can just as readily store and control text-type data such as user guide instructions or design specifications [Ro75]. In addition to access controls SCCS allows the enforcement of

owner access rights, where only a certain identified user of an element stored in the directory is allowed controlled update privileges for that element.

In a software development library, capabilities such as version maintenance, recording of historical change information, and ability to reconstruct previous versions also contribute to control [Be84a]. Such a library facilitates public, visible, non-redundant storage of information [Ta77]. For example, a well-managed and controlled library of specifications and reusable code is essential in the Fifth Generation techniques [Ro84].

Seagraves and Sagan [Se81] have described a library management system for Software Configuration Management as having the three functions of Source Management, Configuration Management and Change Management. Source Management includes capturing and cataloging source items and changes to them, as well as controlling access. Configuration Management captures and catalogs derived items, identifies product revisions, and provides release control. Change Management captures problem reports, change requests and test data and outputs, and tracks changes.

1.2.3.4 Modification Request and Problem Reporting Systems

Since Configuration Control involves control of changes, a Modification Request system enables this control by providing a tool for documenting reports of problems or requests to change a software system, during either development or maintenance phases. Therefore, this

Configuration Control mechanism also implements functions of the Configuration Status Accounting task. In addition, a Modification Request system can implement access and update authorization controls, show relationships and dependencies between configuration items and provide project scheduling aids.

One Modification Request system, called the Modification Request Control System (MRCS), has been described by Knudsen, Barofsky and Satz [Kn76]. As defined in this system, a Modification Request (MR) is any request to change the product (application) system. The request could be a trouble report, a request to change design specifications or a request to make an enhancement to the system. The MR system "provides the capability to: (1) interactively create, update, and print MRs; (2) track and record the flow of the MR through the system development cycle; and (3) provide management with timely MR status information via reports and on-line inquiries. MRCS supports many projects, each project with its own MR data base and commands. It provides, via common control logic, standard operations such as the creation, updating, and printing of MRs, and the extraction of data from them." The system allows users to provide feedback to developers, helps management know who is doing what to the product system and enables developers to determine when individual assignments are scheduled for completion. In addition, through the use of SCCS and the recording of the release to which each MR applies, MRCS can be used in the system generation procedure to produce an entire release of the product system.

Another Modification Request system has been implemented as part of the 3B20D Processor Software Development System [Ro83]. This MR system also uses a database to store requests and provide reports. In addition, a separate Change Management System is used to control approvals and control change activity through interfaces with the MR system. For example, a developer cannot modify source until a Modification Request is assigned.

Recording and reporting of problems and change status have also been identified as necessary functions in a set of Project Management tools proposed in a project called STARS [Lu83, Dr83]. This project is part of an effort by the Department of Defense to develop more adaptable and reliable software systems and improve productivity. This goal requires the improvement of the environment in which software is developed and supported. SCM Control and Status Accounting tools for change management and reporting are part of this desired environment.

1.2.3.5 Version Control Systems

Version Control Systems, another Configuration Control mechanism, are systems which control releases of software. They often have the capability to generate a whole software system from specified releases of specific programs. As described in the examples above, some Modification Request systems also implement Version Control functions. Other control tools implement primarily Version Control. One example of this type of system is described by Tichy [Ti79]. This system controls development by ensuring consistency of module interconnections. The

control is accomplished by maintaining consistency of module interfaces when changes are made and by coordinating the selection and generation of versions and configurations. A language called INTERCOL (module INTERCOConnection Language) has been created to describe the interfaces in the subject system. These descriptions aid in identification of what modules will be affected by proposed changes, estimation of their impact and creation of a particular version of the subject system.

The Revision Control System (RCS) is another system which controls implementation of revisions to text and creation of new versions [Ti82]. This system is best described as it contrasts with SCCS. Whereas SCCS stores changes, called deltas, to elements in a first-to-last and merged organization, RCS stores changes in a last-to-first and separate organization, which Tichy calls reverse deltas. The advantage of the latter organization is that the latest version can be accessed faster. However, both SCCS and RCS control access to elements and enable an item to be "locked" when it is retrieved for updating purposes. RCS has been implemented to be compatible with the UNIX tool called MAKE, which enables automatic system regeneration from the latest version of its elements.

1.3 The SCM Control Project in Relation to the Literature

As previously stated, the objective of this project was to develop some automated tools to support control of software development. The system developed in this project implements several of the functions which have been identified in current literature as being vital to the

control of software development. These functions combine both the SCM Configuration Control and Configuration Status Accounting tasks.

The subject system is a Modification Request system similar to the Modification Request Control System described by Knudsen, Barofsky and Satz [Kn76]. Both systems are implemented on a UNIX Operating System and include a type of database for storage of Modification Request information. MRCS implements this database as a regular UNIX file, while this project uses an SCCS file system. Both systems include software to interactively access, update and report on the information in the database. Both systems include a field to indicate status or current state of a Modification Request. The system documented in this report tracks more detailed states than does MRCS.

Most Version Control Systems, such as Tichy's Revision Control System [Ti82], enable restricted access to a configuration item and allow it to be locked if it is being updated. The system developed in this project authorizes update permission to the responsible person when a Modification Request is assigned to be implemented. The system also includes a program to validate MRs when a controlled configuration item is updated. This type of automated control falls within the scope of an "active" SCM DBMS as described by Sibley [Si81].

Most Modification Request and Version Control Systems described in the literature include some Configuration Status Accounting functions. CSA functions control the recording and reporting of events during the project life cycle. These events include the changing of configuration

items. The system developed in this project implements a function to track the states of MRs as well as list certain information contained in Modification Requests. This function includes an option to list all configuration items affected by a specified MR, which could provide cross-reference capabilities to a configuration and baseline storage system. These functions could aid in version or release control as described by Tichy [Ti82] and by Knudsen, Barofsky and Satz [Kn76].

1.4 Summary

As a result of the increasing complexity of computer systems, it is becoming more difficult to develop software on time and within budget and still meet the user's requirements for the product. Software Configuration Management is a discipline that can contribute to the control of software development in order to produce a product with integrity.

It has been observed that, "as a management discipline, the goal of SCM is to promote efficient software development practices by making the results of the development processes visible." However, the SCM tasks themselves in order to function most effectively should, by contrast, not be visible [Sc85].

The state-of-the-art in software development is constantly and rapidly changing, with increasing use of microcomputers, minicomputers and Fifth Generation Systems. In order to be successful and beneficial, the application of Software Configuration Management must also be dynamic. But the concepts of SCM must be applied in a controlled

manner, so that their mechanisms will not hinder the project objectives and result in increased costs, schedule delays and confusion [Ra80].

Just as stated in the discussion by Bersoff, Henderson and Siegel of the Configuration Control engine, the Control and Status Accounting functions and the tools that implement them should not grind the software development process to a halt. The elements of control (i.e., tools, procedures, people and documentation) should not get in the way of the production and maintenance of software which satisfies both user and management requirements. The functions of SCM and the control tools should "make doing the right thing the easiest thing to do [Be84b]."

CHAPTER 2. REQUIREMENTS OF THE SCM CONTROL SYSTEM

2.1 Scope of Implementation

The objective of this project is to develop a set of tools to aid in the control of software development by supporting the Software Configuration Management processes of Configuration Control and Configuration Status Accounting. This project is part of the Kansas State University software development environment prototype. Other projects implement additional tools for this prototype, including Configuration Identification tools. However, the implementation described in this paper was required to be a stand-alone project having no dependencies with any other project.

If the prototype is expanded in the future to provide interfaces between some tools, it is expected that the SCM tools would be the method by which the other tools (such as a Data Dictionary or a Specification Consistency Checker) would store, retrieve, update, track and, most importantly, control their data. This relationship between the SCM tools and the KSU prototype is depicted in Figure 1.

2.2 General Requirements

The general requirements for the control tools include environmental, software, and useability considerations. The tools must be executable on the Kansas State University UNIX Operating System. They must be suitable for inclusion in the software development environment prototype currently being developed at

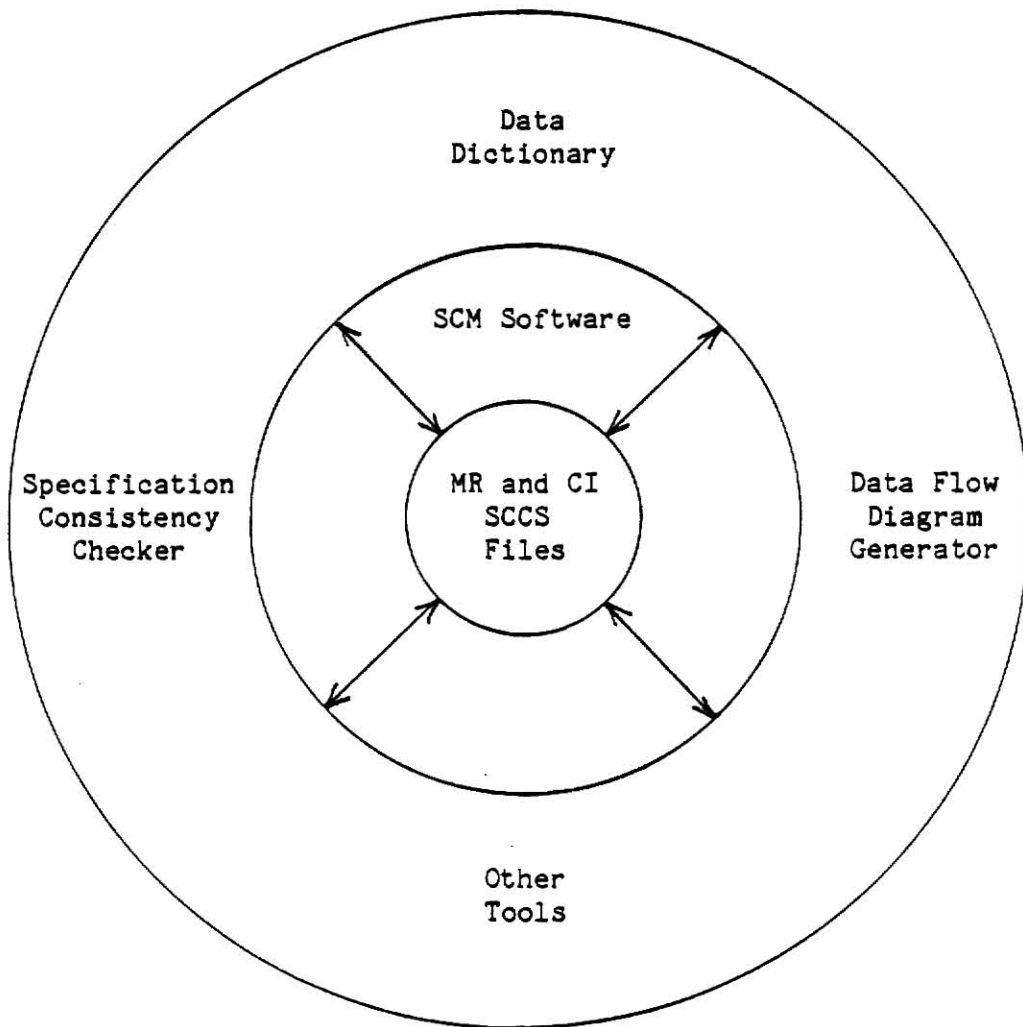


Figure 1. Relationship of SCM to KSU Prototype Tools

Kansas State University. That is, they must be modular, flexible and easy to use to enable compatibility in a research tool environment. The tools must be implemented in some combination of C Language programs, Shell programs, and UNIX utility functions. In addition, the UNIX Source Code Control System (SCCS) software package must be available. Output messages, especially error messages, must be readily

understandable by the user. Any internal error messages must be routed to a central collection point, to be dealt with by a person knowledgeable about the system software.

2.3 Specific Requirements

As the literature on Software Configuration Management indicates, a Modification Request System is required for effective control of software development. Therefore, this project will implement such a system. The system must be useful in any phase of system development, from the initiation and developmental phases through maintenance. Modification Requests are required to be applicable to any text oriented configuration item (e.g., requirements specifications, design documents, source code or user instructions). The capability must exist to maintain a history of Modification Requests (i.e., prior versions). It is necessary that the content of MRs be controlled, especially to ensure that required information is present. Status information must be retained on the MRs to provide tracking capabilities.

2.4 Functional Requirements

The MR System must be able to support submission of a request to modify the configuration (i.e., a Modification Request), to update a Modification Request, to check for valid Modification Requests, to list Modification Request contents, to obtain a cross-reference between configuration items and Modification Requests (one CI may be changed by multiple MRs and one MR may affect multiple CIs), to track Modification Requests by status and to provide some control over updates to

configuration items, such as the ability to authorize a person to update a configuration item. The submission, updating and control capabilities must be restricted to a specific person or persons who are authorized to perform those functions. The following functions must be available to the user:

1. Submit a Modification Request — This function will prompt the user for required and optional fields and create a Modification Request. The MR will be stored in a controlled file.
2. Update a Modification Request — The user will be prompted for the fields to be updated. The program will check the input for consistency. If the user is updating information concerning configuration items affected by the Modification Request, the program will authorize update of those items.
3. Check Modification Requests — This is a validation function. When update is attempted for certain controlled configuration items, this function can be executed to check the list of MRs supplied by the configuration item update program, to ensure that the MRs exist. In addition, this function can be executed by the user to check for existence of MRs.
4. Select Modification Requests — This program will select all MRs that meet certain criteria, specifically a particular status code, supplied by the user. A list of all selected MRs will be created in a file, for potential input to other functions.

5. List a Modification Request -- Given a Modification Request number supplied by the user or from the Select MR function, this program will list the specified contents of the MR. The user will be prompted to identify the information to be listed, including all information in the MR, the description field, status information, and the configuration items affected by the MR.

2.5 Design Decisions Based on the Requirements

One of the requirements of this project is that it be implemented on a UNIX Operating System. Therefore, it was decided to implement the Modification Request database as an SCCS directory/file system to enable use of existing storage, retrieval and access control and validation capabilities, as well as the maintenance of history information in the form of deltas. These capabilities are primitive functions of SCCS, and can be accessed through C language or Shell programs. In addition, SCCS can store any text-oriented data, which would include MR data.

For ease of use, each function will be implemented as an individual C language program. Modular design will be accomplished by developing separate subprograms for subfunctions of the main MR System functions. In addition, a shell will be developed to perform a supervisory function and "call" the main function programs as specified by user request.

CHAPTER 3. DESIGN OF THE SCM CONTROL SYSTEM

3.1 Overview

This chapter describes the specifications developed during the design of the tools to support the Software Configuration Management Control System. Figures 2 and 3 show high-level Data Flow Diagrams of the functions, people, data, and data stores which make up and interact with the Modification Request System to be developed in this project. An overview Hierarchy Diagram of the programs and subprograms in the Modification Request System is displayed in Figure 4.

The Data Flow Diagrams display the functions of the MR System as stated in the requirements. Certain functions are triggered by input from a person described as the Modification Request Administrator (MR Administrator). This person may be responsible for installing the MR System software. The MR Administrator for a particular system/project is responsible for maintaining certain controls over the contents of MRs stored for that system. These controls are implemented by restricting the submit and update functions to the MR Administrator. That person will get input from the user of the application system, project management and developers, which is used to submit and update MRs stored in the MR System.

The other functions of select, list and validate can be performed by any user of the MR System (e.g., a user of the application system, the MR Administrator, Project Management and developers). In addition, a subfunction of validate, called Check MR, can be performed by a

special external system. This system would be any implemented Configuration Item Identification/Storage system that uses SCCS to store configuration items. The Check MR function would be called by the Configuration Item system when a request is made by someone to change an item and that person has specified what MR(s) are associated with the change to the item.

The Data Flow Diagrams in Figures 2 and 3 show some special data stores used by the MR System. For control purposes, it is necessary to retain the numbers of the first and last MRs that are created. Additional controls require that the login id of the person authorized to create and change MRs (i.e., the MR Administrator) be available. Some of the SCCS primitives accessed by the MR System require that the contents of a Modification Request be stored in a separate area. This data store is labelled hold.mrrec in Figures 2 and 3. Lastly, the standard error (std.err) data store will contain detailed internal error messages which might be generated by the MR functions.

As shown in the Hierarchy Diagram in Figure 4, the MR System will contain a shell to interact with the user and to act as a supervisor to "call in" the MR functions described in the system requirements. Each function will be a separate, modular program. Programs which perform primitive tasks or often used subfunctions will be written as subprograms and will be called by the main programs. SCCS functions will be used as "called" primitive subfunctions of the main programs or subprograms.

In this chapter, the MR System Primary Shell is described first, followed by specifications for each main function. Then each subfunction is documented. The remainder of the chapter contains specifications for the major data structures used in the system.

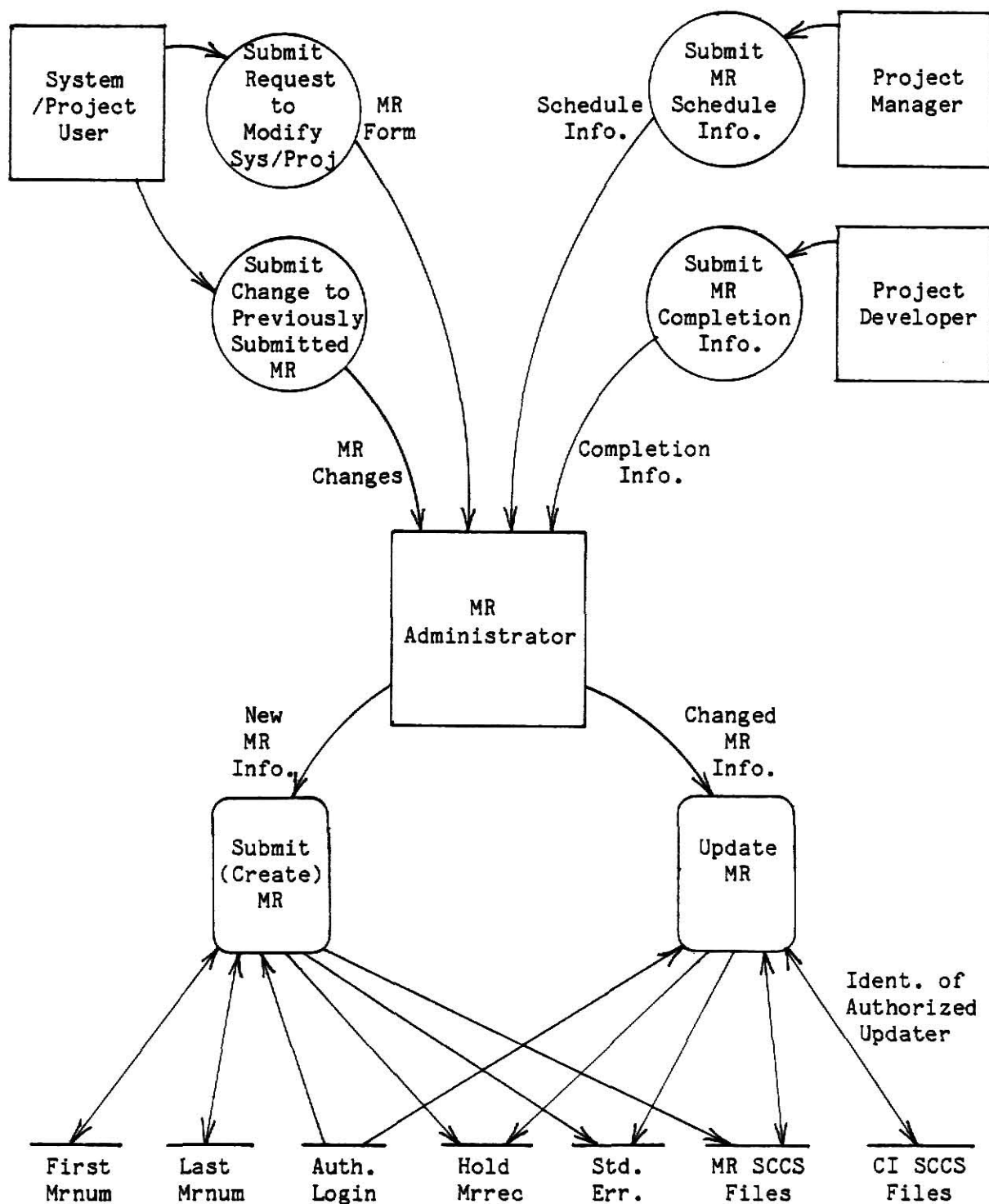


Figure 2. Data Flow Diagram of MR System (Submit and Update)

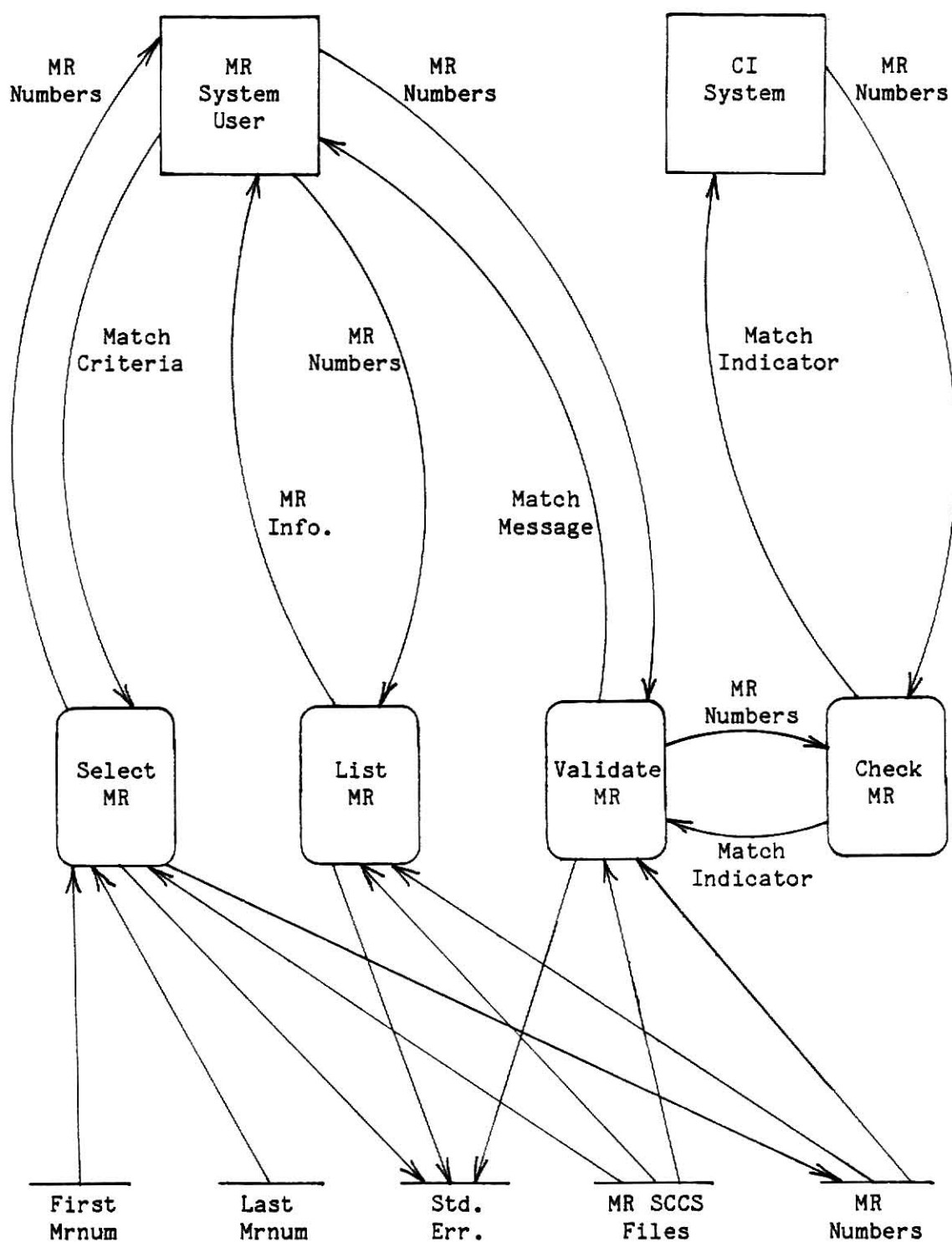
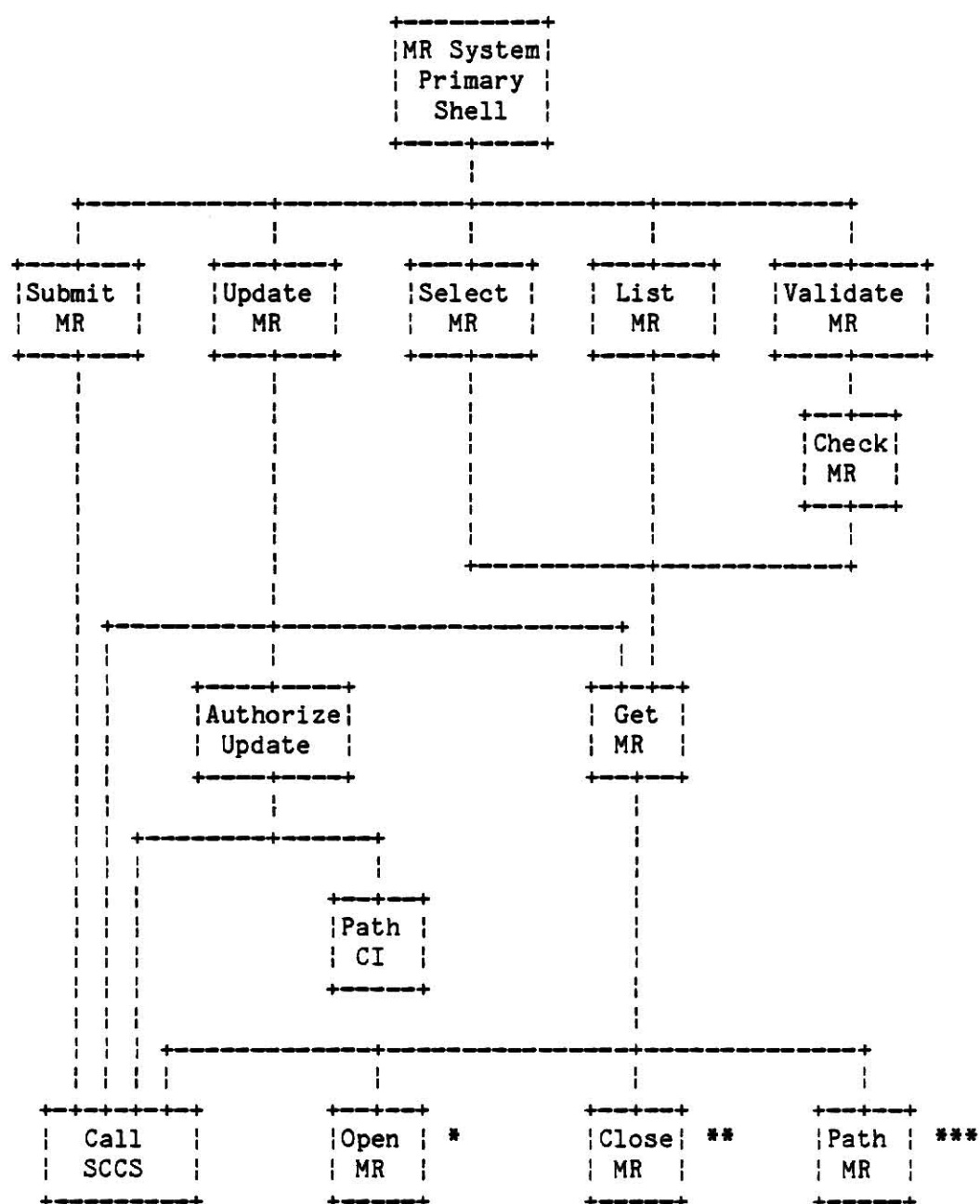


Figure 3. Data Flow Diagram of MR System (Select, List and Validate)



- * — also called by submit, update, select, list and check
- ** — also called by submit and update
- *** — also called by submit, update and select

Figure 4. Hierarchy Diagram of Modification Request System

3.2 Primary Shell Specifications

Description:

This shell is a supervisor routine, which executes one or more of the main programs or exits the MR System, depending on options specified by the user.

Input:

- An option from the user to indicate the MR function to be executed

Output:

- A menu of options indicating the MR System function available to the user

3.3 Main Programs

3.3.1 Submit a Modification Request

Description:

This program creates Modification Requests. Its use is restricted to the MR Administrator. It prompts the MR Administrator for the required fields of description, originator's identification and system/project identification. It also prompts the MR Administrator for the optional fields of impact, priority, requested completion date and severity. The program creates a Modification Request number based on current date and the last mrnum file, stores the new number in the last mrnum file, and also stores it in the first mrnum file if this is the first MR for this MR System. The program writes the MR Administrator's input and program supplied information in an intermediate file, sets up the parameters and arguments for an SCCS

"admin" function, and calls SCCS with those arguments. Appropriate error messages are returned to the MR Administrator.

Input:

- MR Administrator input from the terminal
- The first mrnum file
- The last mrnum file
- The file containing the authorized mradmin login id

Output:

- A Modification Request SCCS file
- The first mrnum file
- The last mrnum file
- The MR record hold file
- Appropriate error or normal termination messages to the MR Administrator
- Error messages to the mradmin standard error file
- A return code indicating normal or abnormal program termination

3.3.2 Update a Modification Request

Description:

This program changes fields in a Modification Request based on identification of particular fields. Its use is restricted to the MR Administrator. Validation of input is performed to ensure that required fields are not nullified and that appropriate fields are consistent. The SCCS "get" function is performed to obtain the MR record. If the MR is being assigned for implementation, the affected configuration item fields are populated and the identification of the

implementer(s) is added to the list of persons authorized to update the configuration item. The implementer is the person (or persons) on the project team who will change the configuration item(s) affected by this MR. The implementer's identification is also stored in the MR. If the MR is being updated to indicate implementation complete or request is closed, the implementer(s) identification is deleted from the list of persons authorized to update the configuration items. This method of retaining implementer identification allows the update ability to be limited to one person or specific people. The parameters and arguments for an SCCS "delta" function are set up and the SCCS programs are called.

Input:

- MR Administrator input from the terminal
- The file containing the authorized mradmin login id
- The Modification Request SCCS file

Output:

- The MR SCCS file
- The MR record hold file
- The Configuration Item SCCS file affected by the MR
- Appropriate error or normal termination messages to the MR Administrator
- Error messages to the mradmin standard error file
- A return code indicating normal or abnormal program termination

3.3.3 Select Modification Requests

Description:

The program selects all Modification Requests that meet certain criteria, specifically certain status codes. The user is requested to enter the desired status code. Then the program loops through all MRs, beginning with the MR number stored in the first mrnum file through the MR number stored on the last mrnum file. Each MR is tested for the requested status. Selected MR numbers are written to a file in the user's directory and optionally displayed on the terminal.

Input:

- User input from the terminal
- The first mrnum file
- The last mrnum file
- All MR SCCS files

Output:

- A file containing selected MR numbers
- Optional terminal display of selected MR numbers
- Appropriate error or normal termination messages to the user
- Error messages to the mradmin standard error file
- A return code indicating normal or abnormal program termination

3.3.4 List a Modification Request

Description:

This program lists Modification Requests depending on user options.

A list can consist of all fields in the MRs specified, or all affected configuration items, status information or description. The MR numbers can be entered as standard input from the terminal or read from a file.

Input:

- User input from the terminal
- An optional file of MR numbers
- The specified MR SCCS files

Output:

- The requested contents of the specified MRs
- Appropriate error or normal termination messages to the user
- Error messages to the mradm standard error file
- A return code indicating normal or abnormal program termination

3.3.5 Validate Modification Requests

Description:

This program validates Modification Requests by calling the Check MR subprogram to check that the MRs exist. The MR numbers can be entered as standard input from the terminal or read from a file.

Input:

- User input from the terminal
- An optional file of MR numbers
- The specified MR SCCS files

Output:

- Appropriate error or normal termination messages to the user
- Error messages to the mradm standard error file

- Return code indicating normal or abnormal program termination

3.4 Subprograms

3.4.1 Check Modification Requests

Description:

This subprogram checks for valid MR numbers. When passed a list of MR numbers, it will try to get each MR in the list. The MR will not actually be retrieved, only checked for existence. This subprogram can be called by the Validate MR main program or by a program that uses the SCCS "delta" function to change a configuration item. The latter program can require that MR numbers be entered when changing an item. In that case, SCCS can be told to execute this Check MR subprogram to check for existence of those MRs.

Input:

- A list of MR numbers separated by blanks and/or tab characters and terminated with a new-line character

Output:

- Error messages to the mradmin standard error file
- A return code of 0 if all numbers were valid, -1 if any were invalid

3.4.2 Authorize Update of a Configuration Item

Description:

This subprogram will authorize a person to update a specified configuration item. The use of this subprogram is restricted to the MR Administrator. Given a specific configuration item and login id,

the subprogram will set up arguments and call the SCCS "admin" primitive to add or delete the id in the list of authorized users for the item.

Input:

- A configuration item identification code or name
- The login id of the person to be authorized or de-authorized
- A flag indicating if authorization is to be added or deleted
- The identified Configuration Item SCCS file

Output:

- The Configuration Item SCCS file
- Error messages to the mradmin standard error file
- A return code of 0 on normal termination and -1 if there were any errors

3.4.3 Get a Modification Request Record

Description:

This subprogram reads the MR SCCS files. Given a specific MR number, the program will retrieve the MR record as read-only or with the ability to change and replace the record, or will suppress actual retrieval of the record. If the record is retrieved, it will be stored in a global structure. When the SCCS "get" primitive is executed, it puts the record in the user's current directory. This Get MR subprogram will delete it from the user's directory and, if it is to be retrieved for update, it will be written to the hold mrrec file in the mradmin directory.

Input:

- An MR number
- The MR SCCS files
- A flag indicating if actual retrieval is to be suppressed, if the MR is to be retrieved for subsequent update, or if it is to be retrieved as read-only

Output:

- The MR record structure if retrieval is not suppressed
- The hold mrrec file in the mradmin directory if the MR is to be updated
- Error messages to the mradmin standard error file
- A return code of 0 on normal termination and -1 if there were any errors

3.4.4 Setup Path of CI Files

Description:

This subprogram sets up a variable with the full pathname of the directory under which the CI files for the project are stored. The default path has the CI directory under the ciadmin directory which is under the same directory as the users' login directories. If some other structure is used, the subprogram requests the user to enter the correct path to the CI directory.

Input:

- User's login pathname
- Pathname of CI directory entered by user if not default structure

Output:

- Variable containing full pathname of CI directory

- A return code of 0 on normal termination, -1 if there were any errors.

3.4.5 Call an SCCS Function

Description:

This subprogram executes the UNIX system functions of "fork" to create a child process, "execv" to execute the specified SCCS primitive in the child process and "wait" to suspend execution of the parent or calling process until the child process terminates.

Input:

- A series of parameters in the format of the arguments required for the UNIX system "execv" function. The first argument is a pointer to the name of the SCCS primitive to be called (e.g., "admin"). The remaining arguments are pointers to parameters as required by the individual SCCS primitive (e.g., "-n" is one argument passed to the "admin" primitive to create a new SCCS file).

Output:

- Error messages to the mradmin standard error file
- A return code of 0 on normal termination and -1 if there were any errors

3.4.6 Open a File in the MR System

Description:

This subprogram opens the specified file used by the MR System.

Input:

- The name of the file to be opened

- A flag indicating the mode to be used to access the file (i.e., "r" for read and "w" for write)

Output:

- Error messages to the mradmin standard error file
- A return code of 0 on normal termination, -1 if there were any errors

3.4.7 Close a File in the MR System

Description:

This subprogram closes the specified file used by the MR System.

Input:

- The FILE pointer to the file to be closed

Output:

- Error messages to the mradmin standard error file
- A return code of 0 on normal termination, -1 if there were any errors.

3.4.8 Setup Path of MR Administrator

Description:

This subprogram sets up a variable with the full pathname of the MR Administrator's mradmin directory. The default path has the mradmin directory under the same directory as the users' login directories. If some other structure is used, the subprogram requests the user to enter the correct path to the mradmin directory.

Input:

- User's login pathname
- Pathname of mradmin directory entered by user if not default

structure

Output:

- Variable containing full pathname of mradmin login directory
- A return code of 0 on normal termination, -1 if there were any errors.

3.5 MR System Directory and File Structure

The Modification Request System has some flexibility as well as dependence in its directory structure. A sample structure is shown in Figure 5. Any number of levels of directories can exist in place of the "project" directory. However, all directories for users of the MR System for a particular project must be under the same project directory as the directory for the MR Administrator. This implies that multiple MR Systems can exist, depending on the number of projects which require the systems.

The MR Administrator directory must be the "HOME" directory for the login-id of the person who has the responsibility for that function. Under that directory will be a subdirectory and five files. The subdirectory (modreq) will contain an SCCS file for each Modification Request. The SCCS file will contain some SCCS-related data and the MR Record. The five files contain the login-id for the person who is the MR Administrator (auth.login), the first MR number assigned for this MR System (first.mrnum), the last MR number assigned for this MR System (last.mrnum), a file which will be used as an intermediate location for the MR being created or updated by the SCCS primitives (hold.mrrec), and

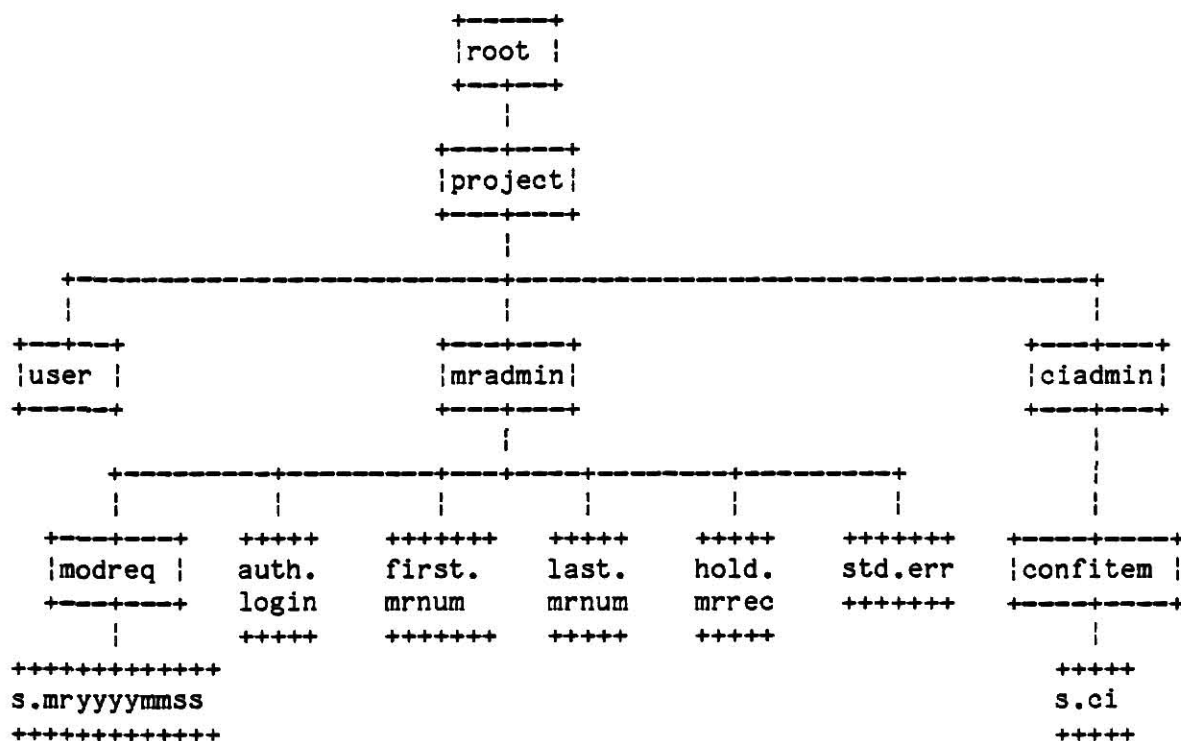


Figure 5. Directory/File Structure of Modification Request System

a file which will contain internal error messages which may be created when a user executes some MR System function (std.err).

Another directory called ciadmin will exist for each project and will be assigned to the Configuration Item Administration function. Under this directory will be a subdirectory (confitem) which will contain the SCCS files where the Configuration Items are stored. This structure is also depicted in Figure 5.

3.6 Modification Request Record

A Modification Request record is a series of fields of the form

```
keyword:value\n
keyword:value\n
.
.
.
keyword:value\n
```

where \n is a newline character.

Each MR record is stored in a unique SCCS file. All keywords shown in the table below exist in the MR record. However, optional or unspecified (depending on status) values will be null. Figures 6a and 6b show information about a Modification Request record. The first column gives the keyword for each field in the record. The last column describes the field. The center column shows a code which indicates if a value for the field is required, optional or computed by a program.

The valid codes are:

```
n = required for new MRs
i = required for status of assigned for investigation
a = required for status of approved for implementation
r = required for status of rejected
d = required for status of deferred for later investigation
g = required for status of assigned for implementation
m = required for status of implementation complete
c = required for status of closed
o = optional
p = computed by program
```

<u>Keyword</u>	<u>Code</u>	<u>Description</u>
mrnum	p	Modification Request number Format is mryyyymmss, where mr = the characters mr yyyy = the year the MR was created mm = the month the MR was created ss = a sequence number within month (01-99)
desc1	n	Description of modification requested, including problem or condition which prompted request
desc2	o	Continuation of description
desc3	o	Continuation of description
desc4	o	Continuation of description
desc5	o	Continuation of description
origid	n	Identification of originator of request
sysprojid	n	Name of system/project primarily affected by request
impact1	o	Impact of request, including benefits, extent of effects on other systems/projects
impact2	o	Continuation of impact
impact3	o	Continuation of impact
impact4	o	Continuation of impact
impact5	o	Continuation of impact
prio	o	Priority of request (relative rank of this request within the group of all requests of the same severity)
rcompdt	o	Requested completion date
sever	o	Severity of problem or condition identified in the request (i.e., system will not run, a major function of the system is significantly inhibited, problem can be circumvented with minor inconveniences, problem is not critical and can be circumvented indefinitely, etc.)
status	p	Status of MR record Normal sequence of status: -> d n -> i -> (one of) -> a -> g -> m -> c -> r
invid	i	Identification of person who is assigned to investigate the request
invdt	p	Date request was assigned for investigation
appid	a	Identification of person who approved request for implementation
appdt	p	Date request was approved

Figure 6a. Contents of a Modification Request Record

<u>Keyword</u>	<u>Code</u>	<u>Description</u>
rejld	r	Identification of person who rejected request
rejdt	p	Date request was rejected
rejreas1	r	Reason request was rejected
rejreas2	o	Continuation of reject reason
rejreas3	o	Continuation of reject reason
rejreas4	o	Continuation of reject reason
rejreas5	o	Continuation of reject reason
defld	d	Identification of person who deferred request
defdt	p	Date request was deferred
defreas1	d	Reason request was deferred (including time frame for reconsideration and reinvestigation of request)
defreas2	o	Continuation of defer reason
defreas3	o	Continuation of defer reason
defreas4	o	Continuation of defer reason
defreas5	o	Continuation of defer reason
asgnto1	g	Identification of person who is assigned to implement the request
asgnto2	o	Ident. of another person assigned to implement
asgnto3	o	Ident. of another person assigned to implement
asgnto4	o	Ident. of another person assigned to implement
asgnto5	o	Ident. of another person assigned to implement
asgnto6	o	Ident. of another person assigned to implement
asgnto7	o	Ident. of another person assigned to implement
asgnto8	o	Ident. of another person assigned to implement
asgnto9	o	Ident. of another person assigned to implement
asgndt	p	Date request was assigned for implementation
targdt	g	Target date for completion of implementation
compdt	p	Date implementation was completed
compdesc1	m	Description of completed implementation
compdesc2	o	Continuation of completed implementation desc.
compdesc3	o	Continuation of completed implementation desc.
compdesc4	o	Continuation of completed implementation desc.
compdesc5	o	Continuation of completed implementation desc.
ci1	m	Configuration Item affected by request
ci2	o	Another configuration item affected by request
ci3	o	Another configuration item affected by request
ci4	o	Another configuration item affected by request
ci5	o	Another configuration item affected by request
ci6	o	Another configuration item affected by request
ci7	o	Another configuration item affected by request
ci8	o	Another configuration item affected by request
ci9	o	Another configuration item affected by request
closld	c	Identification of person who closed request
closdt	p	Date request was closed

Figure 6b. Contents of a Modification Request Record (cont.)

CHAPTER 4. IMPLEMENTATION OF THE SCM CONTROL SYSTEM

This chapter discusses some details of the Modification Request System implemented in this project. The implementation was completed at Kansas State University on the Computer Science Department's Perkin-Elmer 8/32 machine using a Berkeley revision of Version 7 of the UNIX Operating System. The MR system consists of one shell program (mrsys), five C language main programs (submitmr.c, updatemr.c, selectmr.c, listmr.c, and valdatmr.c), five corresponding executable object modules (submitmr.x, updatemr.x, selectmr.x, listmr.x, and valdatmr.x), eight C language subprograms (checkmr, authupdt, getmr, pathci, callscs, openmr, closemr and pathmr) and a file containing a collection of global C language #define's and declarations (mr.global.defs).

Implementation of the functions was performed using a "sandwich" technique. That is, low-level modules such as those which perform input, open, close and call SCCS primitives were coded and tested first. These were followed by implementation of the remainder of the modules using a top-down technique.

The shell, programs and subprograms were written in a modular fashion with a minimum of coupling between modules. Comments were included to explain the functions performed by corresponding sections of code. Variables were defined to contain constants that were used in multiple programs. These definitions are contained in one file so they will be easier to find and change if necessary. The file is included in the main programs by using the statement


```
#include "mr.global.defs"
```

The shell, programs, subprograms and included file contain a total of 2911 lines of code, consisting of executable code and comments and formatting statements to aid in readability and maintenance. The source listings for the shell, programs, subprograms and included file are contained in Appendix D.

The shell program is used as the primary external interface between the user and the MR System. However, each main program may be executed independently of the shell. One use for independent execution is in the selection of multiple Modification Requests with different selection criteria. Each execution of the selectmr program will produce a unique file in the user's directory containing the selected MR numbers. The name of this file will be generated by the UNIX software and will be of the form

```
mrnumXNNNNN
```

where mrnum is the characters mrnum, X is a letter or number and NNNNN is a five-digit number. Several selection files can be concatenated and used as input to an execution of the listmr program.

In addition to performing the front-end interface function, the shell also fulfills the requirement to route internal error messages to the MR Administrator. It does this routing by setting up access to a file under the mradmin directory. Any output from any program, subprogram or SCCS primitive which would normally be displayed on the stderr file at the terminal is then redirected to that mradmin file. External error messages meaningful to the user continue to be displayed

on the stdout file at the terminal.

The callscs subprogram is the interface between the MR System and the SCCS primitive functions of "admin," "get" and "delta." Input to callscs is a series of arguments which will contain the SCCS function to be accessed, as well as arguments to be passed to that function. In order to call the SCCS function, a UNIX system call to "fork" is executed to create a child process. Then a UNIX system call to "execv" transforms the child process into a process to execute the specified SCCS function. Finally, the parent process (i.e., the callscs subprogram) uses the UNIX system "wait" function to wait until the child process is terminated.

By default, the users of a Modification Request System are assumed to have login directories under the same directory as the login directory of the MR Administrator and the CI Administrator. Under the Administrators' login directories are directories where the Modification Request files and the Configuration Item files are stored. If the structure of the particular system/project is different from the default for the MRs or the CIs or both, the subprograms pathmr and pathci and the shell mrsys must be modified. The modifications would consist of commenting out the code which currently executes and removing comment characters from the supplied code which requests that the user enter the full pathname. For example, the users of a particular system of CIs and the MR System for those CIs might have login directories under /sys.1/proj.1, the MR Administrator might have a login directory under /sys.1/mr.data, and the CIs for the same project might be stored under a

directory called /sys.1/ci.sys/proj.1. Pathmr and mrsys (unmodified) would set up a default pathname of /sys.1/proj.1/mradmin. Pathmr and mrsys (modified) would require the user to enter /sys.1/mr.data. Pathci (unmodified) would set up a default pathname of /sys.1/proj.1/ciadmin/confitem. Pathci (modified) would require the user to enter /sys.1/ci.sys/proj.1. Requiring the user to enter the actual pathname if the default cannot be used, rather than coding specific pathnames in the programs, enables the MR System software to be used by multiple projects.

Appendix A contains instructions for installation of the MR System. A User Guide for the MR System can be found in Appendix B.

CHAPTER 5. CONCLUSIONS

5.1 Evaluation of the Implementation

The Software Configuration Management Control System, implemented as a Modification Request System in this project, can be a useful tool for control of the development of software systems. It implements several functions of the Software Configuration Management tasks of Configuration Control and Configuration Status Accounting.

The SCM Control System was required to be compatible with the prototype software development environment being developed on a UNIX Operating System at Kansas State University. The automated tools implemented in the Control System had to be easy to use and maintain. The requirements for the implemented functions included a Modification Request System to create, update, select and list MRs submitted for the purpose of requesting changes or enhancements to configuration items. Also required was the ability to ensure authorized updating of configuration items. These requirements have all been met by the MR System.

The MR System facilitates control over software development by providing control and reporting capabilities on MRs and, to a certain degree, on configuration items. There are also controls over the use of the MR System itself. While some functions such as selection and listing are available to any user within the project, other functions such as MR submission, MR update and authorization of configuration item update are restricted to the MR Administrator.

5.2 Possible Extensions

The effort was made in this project to develop a thorough and useful set of tools to support control of software development. If extensions or modifications to the SCM Control System were to be implemented in the future, the following suggestions are offered:

- provide additional links to a Configuration Item/Baseline identification and storage system that is implemented using SCCS files

(One method of interfacing the MR System and a Configuration Item/Baseline Identification System developed by William H. Wilson, IV [Wi84] is described in Appendix C.)

- extend the MR selection function:

- to allow selection on fields other than status
- to allow user defined selection criteria; the user could define what keyword or combinations of keywords are to be used for a particular selection

- include additional checking by the Modification Request checker function:

- ensure that certain information or approvals are recorded before a configuration item is updated
- ensure that the affected configuration item has been recorded in the Modification Request before the item is updated

• provide additional Status Accounting reports:

- a listing of persons authorized to update configuration items
- Project Management reports for scheduling and project tracking, such as all MRs with a requested completion date in a certain range or all MRs assigned to project team members
- a cross-reference of all MRs applied to specified configuration items and/or baselines; this listing could currently be implemented by scanning the entire MR file, but it would be more efficient to retrieve specified items from a Configuration Item/Baseline identification and storage system to obtain a list of applied Modification Requests
- identification of outstanding (retrieved with intention to update) configuration items and who retrieved them
- a comparison between the current baseline and the last or a specified baseline

REFERENCES

- [Be80] Bersoff, Edward H.; Henderson, Vilas D.; and Siegel, Stanley G. Software Configuration Management: An Investment in Product Integrity. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1980.
- [Be81] Begley, Al. "The GTD-5 EAX Software Development Environment." GTE Automatic Electric Journal, 19, No.6 (November-December 1981), pp.193-198.
- [Be84a] Bersoff, Edward H. "Elements of Software Configuration Management." IEEE Transactions on Software Engineering, SE-10, No.1 (1984), pp.79-87.
- [Be84b] Berlack, Ron; Knirk, Dwayne; Poston, Robert; and Tice, George. Student Guide -- IEEE Standards Seminar on Software Configuration Management. Oakhurst, NJ: Programming Environments, Inc., 1984.
- [Br79] Brooks, Frederick P., Jr. The Mythical Man-Month. Reading, Mass.: Addison-Wesley Publishing Company, 1979.
- [Br80] Bryan, William; Chadbourne, Christopher; and Siegel, Stan; eds. Tutorial: Software Configuration Management. New York: IEEE Computer Society Press, 1980.
- [Br82] Bryan, William L.; Siegel, Stanley G.; and Whiteleather, Gary L. "Auditing Throughout the Software Life Cycle: A Primer." Computer, 15, No.3 (1982), pp.57-67.
- [Br84] Branstad, Martha, and Powell, Patricia B. "Software Engineering Project Standards." IEEE Transactions on Software Engineering, SE-10, No.1 (1984), pp.73-78.
- [Ch76] Chen, Peter Pin-Shan. "The Entity-Relationship Model -- Toward a Unified View of Data." ACM Transactions on Database Systems, 1, No.1 (1976), pp.9-36.
- [De80] Dean, William A. "Why Worry About Configuration Management?" In Tutorial: Software Configuration Management. Edited by William Bryan, Christopher Chadbourne, and Stan Siegel. New York: IEEE Computer Society Press, 1980.
- [Dr83] Druffel, Larry E.; Redwine, Samuel T., Jr.; and Riddle, William E. "The STARS Program: Overview and Rationale." Computer, 16, No.11 (1983), pp.21-29.
- [Du82] Dunn, Robert, and Ullman, Richard. Quality Assurance for Computer Software. New York: McGraw-Hill Book Company, 1982.

- [Es84] Estublier, J.; Ghoul, S.; and Krakowiak, S. "Preliminary Experience with a Configuration Control System." Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, (April 1984), pp.149-156.
- [Ev83] Evans, Michael W.; Piazza, Pamela H.; and Dolkas, James B. Principles of Productive Software Management. New York: John Wiley & Sons, 1983.
- [Fe83] Feigenbaum, Edward A., and McCorduck, Pamela. The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World. Reading, Mass.: Addison-Wesley Publishing Company, 1983.
- [Ho82] Howden, William E. "Contemporary Software Development Environments." Communications of the ACM, 25, No.5 (1982), pp.318-329.
- [Hu81] Huff, Karen E. "A Database Model for Effective Configuration Management in the Programming Environment." Proceedings of the Fifth International Conference on Software Engineering, (March 1981), pp.54-61.
- [IE81] IEEE Standard for Software Quality Assurance Plans. New York: The Institute of Electrical and Electronics Engineers, Inc., 1981.
- [IE83a] IEEE Standard for Software Configuration Management Plans. New York: The Institute of Electrical and Electronics Engineers, Inc., 1983.
- [IE83b] IEEE Standard Glossary of Software Engineering Terminology. New York: The Institute of Electrical and Electronics Engineers, Inc., 1983.
- [Ki83] Kim, K. H. "A Look at Japan's Development of Software Engineering Technology." Computer. (May 1983), pp.26-36.
- [Kn76] Knudsen, D.B.; Barofsky, A.; and Satz, L.R. "A Modification Request Control System." Proceedings of the Second International Conference on Software Engineering, (October 1976), pp.187-192.
- [Lu83] Lubbes, H.O. "The Project Management Task Area." Computer, 16, No.11 (1983), pp.56-62.
- [Mc80] McCarthy, Rita. "Applying the Technique of Configuration Management to Software." In Tutorial: Software Configuration Management. Edited by William Bryan, Christopher Chadbourne, and Stan Siegel. New York: IEEE Computer Society Press, 1980.

- [Pr83] Prager, J.M. "The Project Automated Librarian." IBM Systems Journal, 22, No.3 (1983), pp.214-228.
- [Ra80] Raveling, Jerry L. "Meeting the Evolving Micro Requirement." In Tutorial: Software Configuration Management. Edited by William Bryan, Christopher Chadbourne, and Stan Siegel. New York: IEEE Computer Society Press, 1980.
- [Ro75] Rochkind, Marc J. "The Source Code Control System." IEEE Transactions on Software Engineering, SE-1, No.4 (December 1975), pp.364-370.
- [Ro83] Rowland, B.R., and Welsch, R.J. "The 3B20D Processor & DMERT Operating System: Software Development System." The Bell System Technical Journal, 62, No.1 (1983), pp.275-289.
- [Ro84] Romberg, F. Arnold, and Thomas, Alan B. "Reusable Code, Reliable Software." Computerworld, (March 26, 1984), pp.ID/17-ID/26.
- [Sc85] Schwartz, David P. "Summary of Fourth IEEE SCM Working Group Meeting." ACM Software Engineering Notes, 10, No.1 (1985), pp.68-73.
- [Se81] Seagraves, David A., and Sagan, John. "Configuration Management In Large Software Products." International Switching Symposium -- ISS '81 CIC, 3, Session 31B, Paper 3 (September 1981), pp.1-8.
- [Si81] Sibley, Edgar H.; Scallan, P. Gerard; and Clemons, Eric K. "The Software Configuration Management Database." AFIPS Conference Proceedings -- 1981 National Computer Conference, 50 (1981), pp.249-255.
- [Su83] Support Tools Guide -- UNIXTM System. n.p.: Western Electric, 1983.
- [Ta77] Tausworthe, Robert C. Standardized Development of Computer Software: Part 1 - Methods. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1977.
- [Ti79] Tichy, Walter F. "Software Development Control Based on Module Interconnection." Proceedings of the Fourth International Conference on Software Engineering, (October 1979), pp.29-41.
- [Ti82] Tichy, Walter F. "Design, Implementation, and Evaluation of a Revision Control System." Proceedings of the Sixth International Conference on Software Engineering, (September 1982), pp.58-67.
- [Tr82] Treleaven, Philip C., and Lima, Isabel Gouveia. "Japan's Fifth-Generation Computer Systems." Computer, (August 1982), pp.79-88.

- [UN82] UNIXTM System User's Manual -- Release 5.0. n.p.: Western Electric, 1982.
- [Wi84] Wilson, William H., IV. "A Configuration Item and Baseline Identification System For Software Configuration Management." Master's Report, Kansas State University, 1984.

APPENDIX A. MODIFICATION REQUEST SYSTEM INSTALLATION GUIDE

The Modification Request System must be installed on a UNIXTM Operating System having access to SCCS. For each project or system for which a Modification Request System will be installed, a Modification Request Administrator must be appointed. That person must be assigned a login directory in the directory structure of the project or system. The MR Administrator's login directory must be called mradmin. Other users of this project's MR System will have login directories in other branches of that project or system directory structure. Figure 7 depicts an example of this structure. The figure also shows the ciadmin/confitem directory under which Configuration Item files are stored.

If the users of the MR System for a particular project and system of CIs have login directories that are not under the same directory as the login directory of the MR Administrator for that project, then the subprogram pathmr and the shell mrsys will have to be changed. If the users of the MR System for a particular project and system of CIs have login directories that are not under the same directory as the login directory of the CI Administrator for that project, then the subprogram pathci will have to be changed. The code in the subprogram or shell that is currently executed must be commented out. The following code which currently is commented out must have the comment characters removed. This code will then require the user to enter the full pathname of the mradmin directory in pathmr and mrsys, or the directory under which the CI files are stored in pathci.

The executable programs for the MR System must be installed in a UNIX System directory which is executable by anyone who will be using those programs. The directory could be under a system or project directory, a tools directory as shown in Figure 7 or a standard distributed directory like /usr/bin. However, it is recommended that the source code and executable code not be in a standard library in order to facilitate maintenance. When a new standard release of the UNIX Operating System is installed on a subject computer, it often includes new releases of the standard distributed directories. When the system generation is performed, those new directories would replace any existing directories of the same name. Therefore, any data such as the MR System code, which was not part of the standard release, might be lost. An example of an installation directory for the MR System code is shown in Figure 8.

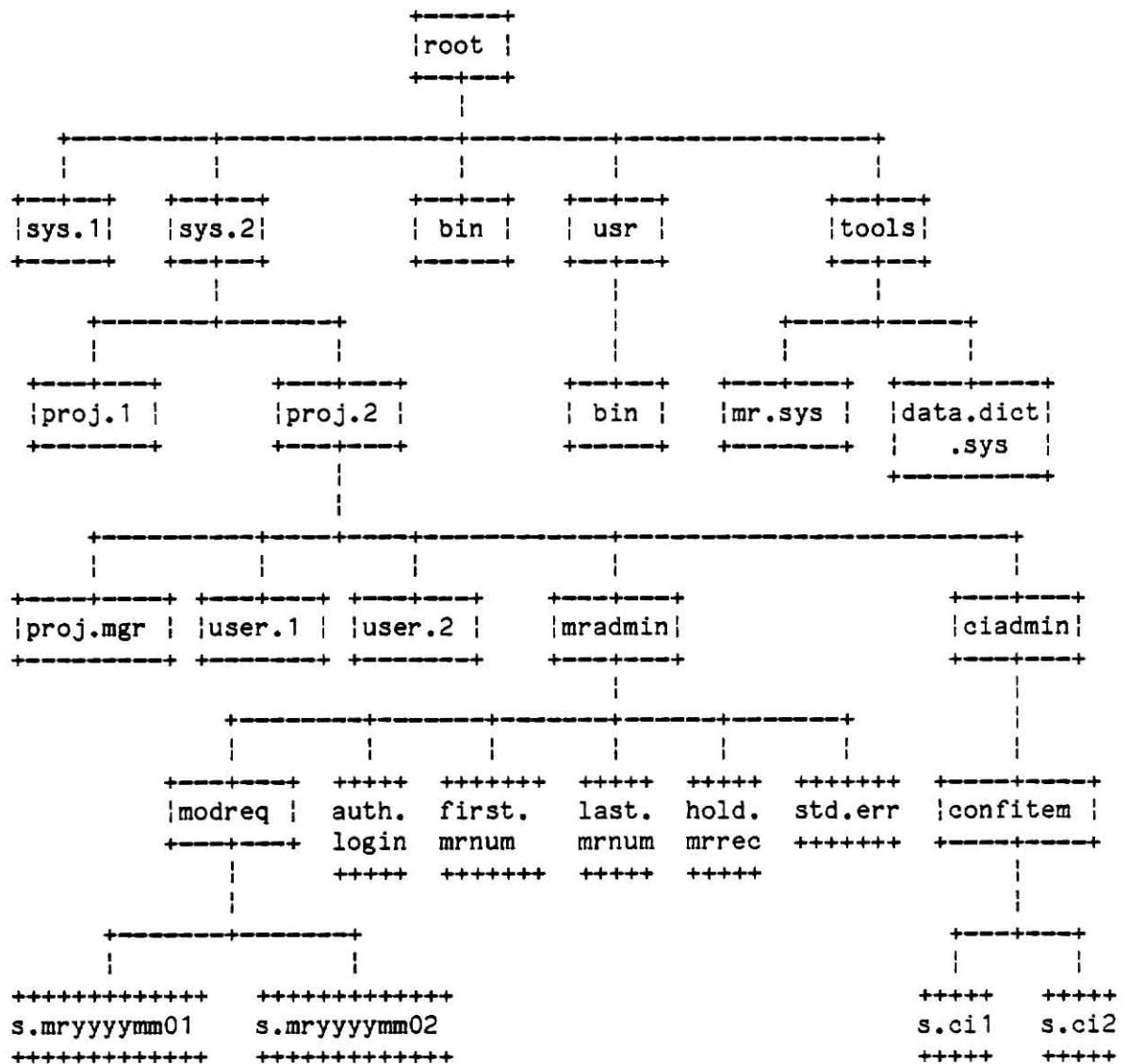


Figure 7. Directory Structure for MR System Installation

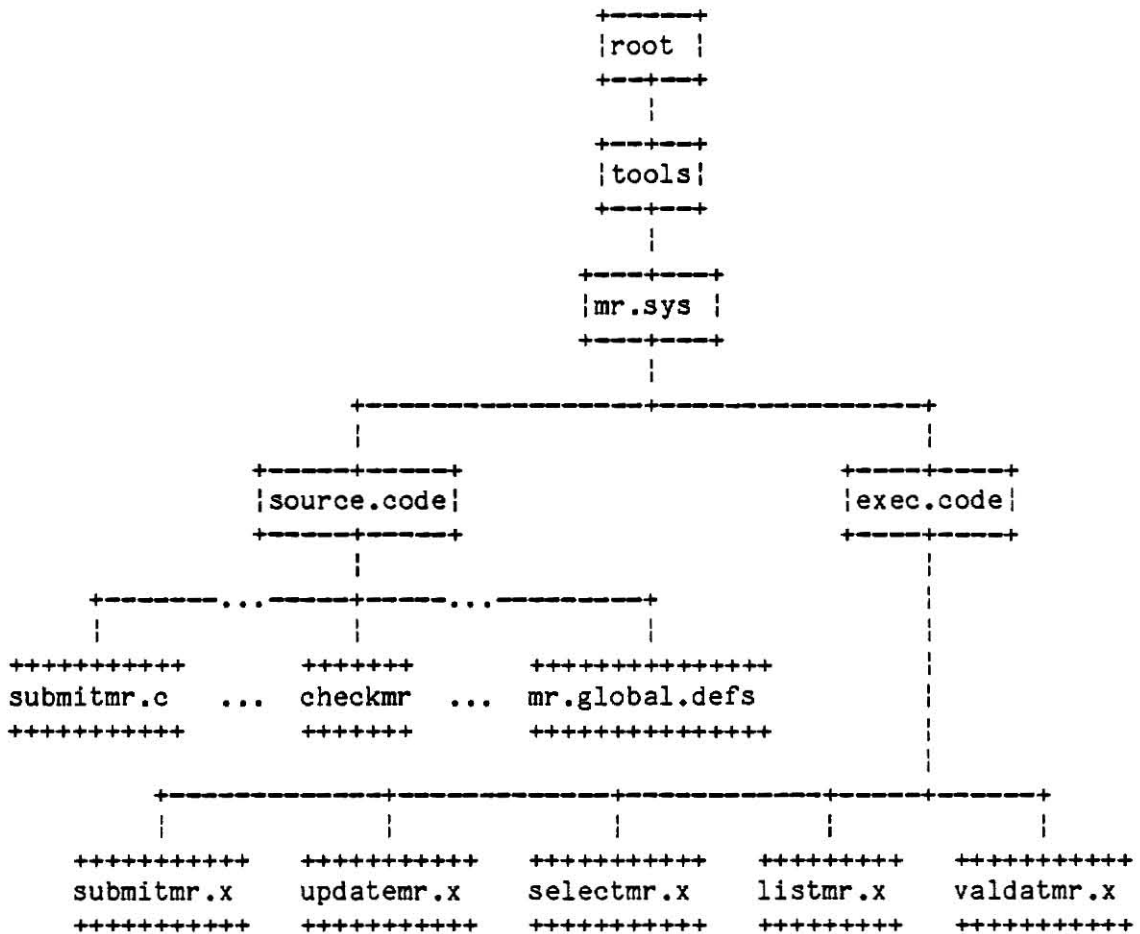


Figure 8. Code Directory Structure for MR System Installation

To install the MR System software, the MR Administrator must use the following procedure:

1. Create directories for the files that will contain the MR System source and executable object modules (e.g., /tools/mr.sys/source.code and /tools/mr.sys/exec.code).
2. Store the source code modules as files in the source.code directory. This directory and these files must have write

permission allowed for the MR Administrator only.

3. Change position to the exec.code directory. Compile each main program (i.e., submitmr.c, updatemr.c, selectmr.c, listmr.c and valdatmr.c) by executing the commands:

```
cc ../source.code/PPPPPPPP.c
```

```
mv a.out PPPPPPPP.x
```

where PPPPPPPP is the name of one of the main programs. This directory and these files also must have write permission allowed for the MR Administrator only.

To install an MR System for a project or system, the MR Administrator must follow these steps:

1. Ensure that the permissions for the mradmin directory are read, write and execute for the MR Administrator and read and execute for everyone else.
2. Make a directory called modreq under the mradmin directory with read, write and execute permissions for the MR Administrator and read and execute for everyone else. This directory will contain the Modification Requests stored as individual SCCS files.
3. Create a file called auth.login under the mradmin directory with read and write permissions for the MR Administrator and read for everyone else. In this file, use the UNIX editor to write one line containing the login id of the MR Administrator.

4. Create a file called first.mrnum under the mradmin directory with read and write permissions for the MR Administrator and read for everyone else. In this file, use the UNIX editor to write one line containing 0000000000 (i.e., ten zeros). The first MR number that is created will replace this line.
5. Create a file called last.mrnum under the mradmin directory with read and write permissions for the MR Administrator and read for everyone else. In this file, use the UNIX editor to write one line containing 0000000000 (i.e., ten zeros). When any MR is created its number will replace this line.
6. Create a file called hold.mrrec under the mradmin directory with read and write permissions for the MR Administrator and read for everyone else. This file will be a temporary storage area for use by SCCS when creating or updating a Modification Request record.
7. Create a file called std.err under the mradmin directory with read and write permissions for everyone. This file will contain standard error messages that describe internal MR System errors.
8. Ensure that a directory called ciadmin has been created in the system/project directory for administration of configuration items. Ensure that the permissions for the ciadmin directory are read, write and execute for the Configuration Item and MR Administrators and read and execute for everyone else.
9. Ensure that a directory called confitem exists under the ciadmin

directory. Ensure that the confitem directory has read, write and execute permissions for the CI and MR Administrators and read and execute for everyone else. This permission allows use of the authupdt subprogram by the Update MR function. The confitem directory will contain the Configuration Items stored as individual SCCS files.

In order to execute the MR System software, each user including the MR Administrator must add the directory where the executable code is stored to the PATH variable in his/her .profile. Using the structure shown in Figure 8, the user would add the following path:

```
/tools/mr.sys/exec.code
```

APPENDIX B. MODIFICATION REQUEST SYSTEM USERS' GUIDE

In order to use the Modification Request System, each user must update the PATH variable in his/her .profile to contain the directory where the executable code for the MR System is stored. The directory name should be available from the UNIXTM System Administrator or the MR System Administrator. Once the .profile is updated, the user can perform functions of the MR System by entering

`mrsys`

This command will begin execution of the MR System Primary Shell. This shell will display a menu of available functions, with a code that the user must enter to select each function or to exit from the MR System. The menu is shown in Figure 9. During execution of each selected option, self-explanatory error and informational messages will be displayed on the user's terminal.

```
*****
*   Modification Request System Primary Function Menu   *
*****
```

The following primary functions are available:

Option	Function
0	Exit the Modification Request System
1	Submit a Modification Request [MR Admin. only]
2	Update a Modification Request [MR Admin. only]
3	Select Modification Requests
4	List Modification Requests
5	Validate Modification Requests

Enter option number for the function you wish to perform:

Figure 9. MR System Primary Function Menu

Submitting Modification Requests

The submit function (option 1 on the Primary Function Menu) will create a Modification Request. This function is available only to the MR Administrator for the project or system. The program will prompt for required and optional information. Up to fifty characters may be entered for each field. The required fields are description, originator's identification and system/project identification. The optional fields are impact, priority, requested completion date and severity. Description and impact may optionally contain up to five lines of text. The message

no id keywords (cm7)

will be displayed when a Modification Request is created. This message is created by the UNIX SCCS software and is informational only.

Updating Modification Requests

The update function (option 2 on the Primary Function Menu) will change information in a Modification Request. This function is available only to the MR Administrator for the project or system. The user will be prompted for the number of the MR to be changed and for the keyword of the field to be changed. A list of keywords can be displayed during execution of the function. The list is shown in Figures 10a and 10b. Up to fifty characters may be entered for each field. Multiple fields can be changed in one execution of the function. If affected configuration item identifications are being entered, the program will accept up to 9 items. If status is being changed along with other information, it should be changed first. All changes entered by the user will be stored in a hold area. The user will be given the option to save those changes in the actual Modification Request record or abort the update (i.e., the changes will not be written on the record).

```
*****
*                MR Fields that can be changed by the user                *
*****
```

In the following list of MR Keywords, the value of Code indicates when the corresponding field is required:

```
n = MR is new
i = MR is assigned for investigation
a = MR is approved for implementation
r = MR is rejected
d = MR is deferred for later investigation
g = MR is assigned for implementation
m = implementation of MR is complete
c = MR is closed
o = optional field
p = initially computed by program; changed by MR Administrator
```

Keyword	Code	Description
desc1	n	Description of modification requested, including problem or condition which prompted request
desc2	o	Continuation of description
desc3	o	Continuation of description
desc4	o	Continuation of description
desc5	o	Continuation of description
origid	n	Identification of originator of request
sysprojid	n	Name of system/project primarily affected by request
impact1	o	Impact of request, including benefits, extent of effects on other systems/projects
impact2	o	Continuation of impact
impact3	o	Continuation of impact
impact4	o	Continuation of impact
impact5	o	Continuation of impact
prio	o	Priority of request (relative rank of this request within the group of all requests of the same severity)
rcomplt	o	Requested completion date
sever	o	Severity of problem or condition identified in the request (i.e., system will not run, a major function of the system is significantly inhibited, problem can be circumvented with minor inconveniences, problem is not critical and can be circumvented indefinitely, etc.)
status	p	Status of MR (computed by submit, changed by update) Normal sequence of status: <div style="margin-left: 100px;"> -> d n -> i -> (one of) -> a -> g -> m -> c -> r </div>

Figure 10a. Keywords Used to Identify MR Fields to be Updated

Keyword	Code	Description
invid	i	Identification of person who is assigned to investigate the request
appid	a	Identification of person who approved request for implementation
rejid	r	Identification of person who rejected request
rejreas1	r	Reason request was rejected
rejreas2	o	Continuation of reject reason
rejreas3	o	Continuation of reject reason
rejreas4	o	Continuation of reject reason
rejreas5	o	Continuation of reject reason
defid	d	Identification of person who deferred request
defreas1	d	Reason request was deferred (including time frame for reconsideration and reinvestigation of request)
defreas2	o	Continuation of defer reason
defreas3	o	Continuation of defer reason
defreas4	o	Continuation of defer reason
defreas5	o	Continuation of defer reason
asgnto1	g	Identification of person who is assigned to implement the request
asgnto2	o	Ident. of another person assigned to implement
asgnto3	o	Ident. of another person assigned to implement
asgnto4	o	Ident. of another person assigned to implement
asgnto5	o	Ident. of another person assigned to implement
asgnto6	o	Ident. of another person assigned to implement
asgnto7	o	Ident. of another person assigned to implement
asgnto8	o	Ident. of another person assigned to implement
asgnto9	o	Ident. of another person assigned to implement
targdt	g	Target date for completion of implementation
compdesc1	m	Description of completed implementation
compdesc2	o	Continuation of completed implementation desc.
compdesc3	o	Continuation of completed implementation desc.
compdesc4	o	Continuation of completed implementation desc.
compdesc5	o	Continuation of completed implementation desc.
ci1	m	Configuration Item affected by request
ci2	o	Another configuration item affected by request
ci3	o	Another configuration item affected by request
ci4	o	Another configuration item affected by request
ci5	o	Another configuration item affected by request
ci6	o	Another configuration item affected by request
ci7	o	Another configuration item affected by request
ci8	o	Another configuration item affected by request
ci9	o	Another configuration item affected by request
closid	c	Identification of person who closed request

Figure 10b. Keywords Used to Identify MR Fields to be Updated (cont.)

Selecting Modification Requests

The select function (option 3 on the Primary Function Menu) will select MRs that meet certain criteria, specifically certain status codes, and will store their numbers in a file in the user's directory. The selected numbers may optionally be displayed at the terminal. The status types that can be selected are shown in Figure 11.

a	=	approved for implementation
c	=	closed
d	=	deferred for later consideration
g	=	assigned for implementation
i	=	assigned for investigation
m	=	implementation complete
n	=	new
r	=	rejected

Figure 11. MR Status Types Available for Selection

Listing Modification Requests

The list function (option 4 on the Primary Function Menu) will display the contents of Modification Requests. The numbers can be entered at the terminal or read from a file. If read from a file, the user will be prompted to enter the file name. In the file, the MR numbers must be stored one per line. The user will also be prompted to enter a code for the type of list desired. Figure 12 shows the information listed for each code.

- a = all fields in the Modification Request
- c = all Configuration Items affected by the MR
- d = the description field
- s = the corresponding id, date and reason fields for
the current status

Figure 12. Codes Used For Listing Modification Requests

Validating Modification Requests

The validate function (option 5 on the Primary Function Menu) will check for existence of specified Modification Requests. The MR numbers can be entered at the terminal or read from a file. If read from a file, the user will be prompted to enter the file name. In the file, the MR numbers must be stored one per line. After the search for the MRs is completed, a message will be displayed which indicates if all specified MRs exist or if some (one or more) were not found.

APPENDIX C. CONFIGURATION ITEM/BASELINE IDENTIFICATION SYSTEM INTERFACE

Software Configuration Management consists of the four tasks of Configuration Identification, Configuration Control, Configuration Status Accounting and Configuration Auditing. The system of tools documented in this report supports the Control and Status Accounting tasks. In order to provide additional support of SCM and augment the capabilities of the Modification Request System, an automated tool for Configuration Identification is required. Such a tool would support the storage, retrieval and update of the Configuration Items to which the Modification Requests apply. A system of tools for this task, called the Configuration Item and Baseline Identification System, has been developed by William H. Wilson, IV [Wi84].

The MR System developed in this project can be easily interfaced with Wilson's CI System in the following manner:

1. When the MR System is installed, the CI System must also be installed according to the instructions in Wilson's paper [Wi84]. The CI directory and files would probably replace the ciadmin/confitem directories and s.ci files documented in this report. However, the projid and projfile directories referenced in Wilson's paper could be called ciadmin and confitem respectively. In addition, the references in the Installation Guide in this report to /sys.2 and /proj.2 could alternately be called /filesys and /scm to correspond to the structure described in Wilson's paper.

2. A copy of the checkmr subprogram must be placed in the same directory as the SCCS software (i.e., the executable versions of "admin", "delta", etc.). This copy allows the CI System to check for valid MRs when a CI is updated.
3. The pathci subprogram might have to be changed, in order to allow the user to enter the pathname of the directory where the CI files are stored. For example, as shown in the CI System documentation, the user might have to enter /filesys/scm/projid/projfile.
4. A sample directory/file structure for the interface of the MR System with the CI System is shown in Figure 13. This structure is project oriented. That is, each project has its own SCM directory. No modifications would be required to the pathci or pathmr subprograms for this structure. An alternate structure is depicted in Figure 14 and is oriented toward SCM. That is, all projects are under one scm directory. This structure would require modification of the pathci subprogram to allow the user to enter the pathname /filesys/scm/proj.1/cibldir. If all projects had a similar structure, where the CI files were always stored in a directory such as ciadmin in Figure 13 or cibldir in Figure 14, and that directory was always located under the same directory as both the users' login directories and the mradmin directory, then the code of pathci could be changed to default to that structure. This change would mean that the users would not have to enter any pathnames. The MR System and CI System could, as a result, still be used by multiple projects.

5. Some controls have been implemented in the CI System, as in the MR System, over who can perform certain operations. Specifically, the CIs are owned by the CI Administrator and only that person can change the update permissions for the CIs. This means that if the MR Administrator does not have the same login id as the CI Administrator, then the authupdt subprogram in the MR System will not be allowed by the CI System to change the CI update permissions. If the MR and CI administrative functions are not performed from the same login id, then only one of the following two changes must be performed. The first change would be made to the MR System and involves commenting out the authupdt calls in the updatemr.c program. As a result, the CI Administrator would have to use the updatusr function in the CI System software to perform the function of authupdt. The second change involves both the CI System and the MR System. This alternative requires that the CI System software be changed to allow multiple login ids to be stored in the /scm/adm file that is created by the install function and checked by the updatusr function. This alternative also involves changing the updatemr.c program in the MR System by replacing the calls to authupdt with calls to the CI System function of updatusr. An explanation of the method that must be used by a C language program to call the updatusr function is given in Chapter 4 (Implementation Details) of Wilson's paper.

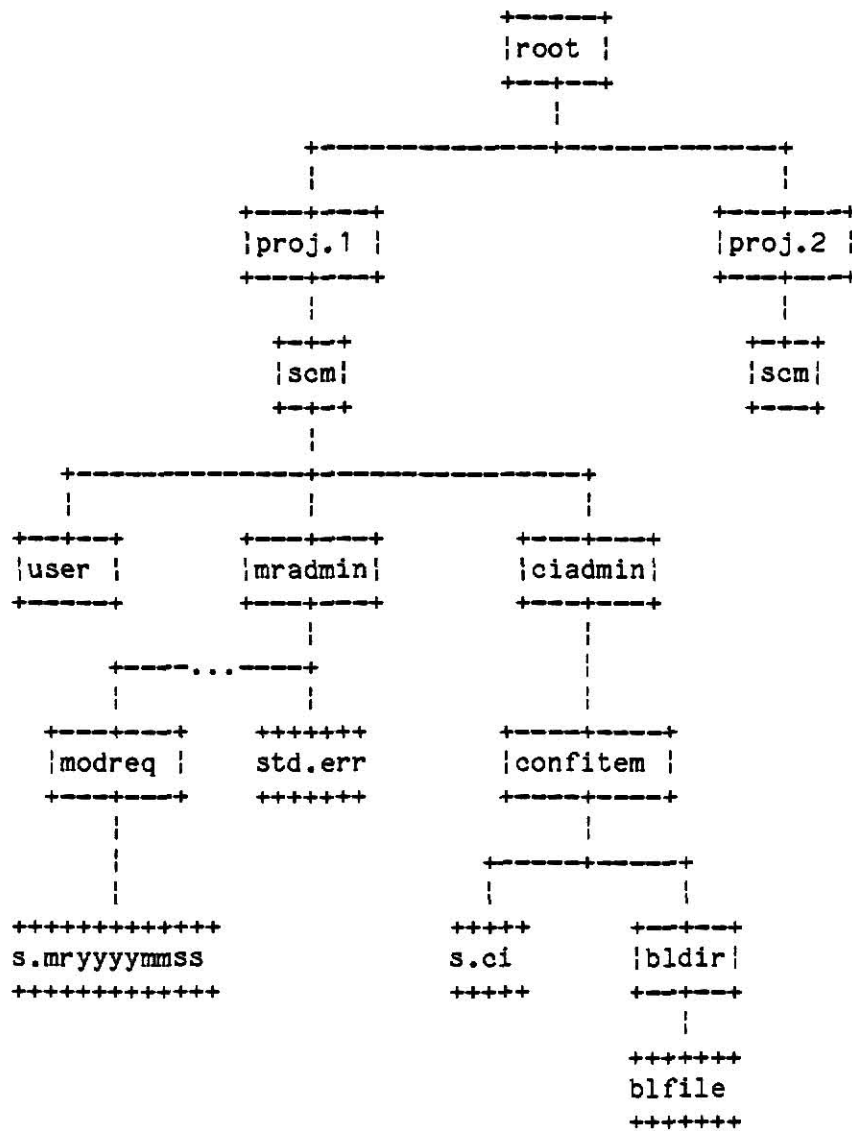


Figure 13. Project Oriented Structure for MR System/CI System Interface

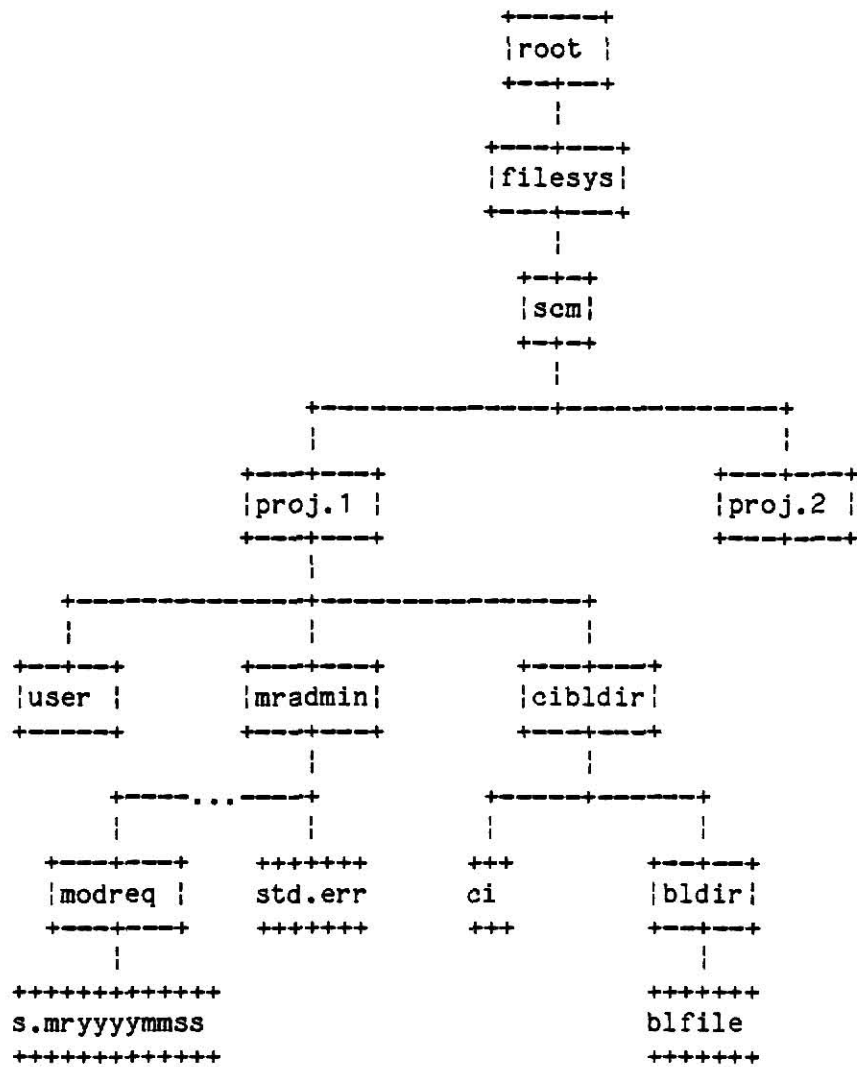


Figure 14. SCM Oriented Structure for MR System/CI System Interface

APPENDIX D. SOURCE CODE

authupdt	80
callsccs	82
checkmr	84
closemr	86
getmr	87
listmr.c	91
mr.global.defs	99
mrsys	100
openmr	102
pathci	103
pathmr	105
selectmr.c	107
submitmr.c	113
updatemr.c	125
valdatmr.c	142


```

/*****
/* authupdt -- authorize update of a Configuration Item */
*****/

authupdt(loginid, ci, action)

char loginid[];
char ci[];
char action[];

    /* loginid = login id of person to be authorized/deauthorized */
    /* ci       = identification of configuration item to be updated */
    /* action   = "a" if add, "e" if delete authorization */
    /* return   = 0 if successful, -1 if unsuccessful */

{
    char *funcargs[4];

    char cidir[100];
    char funcarg0[200];
    char funcarg1[200];
    char funcarg2[200];

    int cidirlen;
    int loginidlen;

    /* get path for ci directory */
    if (pathci(cidir) != 0)
    {
        fprintf(stdout, "Cannot get path for CI directory.\n");
        fprintf(stdout, "Please notify MR System Administrator.\n");
        return(-1);
    }

    /* initialize SCCS function argument areas */
    strncpy(funcarg0, "\0", 200);
    strncpy(funcarg1, "\0", 200);
    strncpy(funcarg2, "\0", 200);

    /* call SCCS admin function to update authorization for the CI */
    strncpy(funcarg0, "admin", 6);
    funcargs[0] = funcarg0;
    strncpy(funcarg1, "-", 2);
    strcat(funcarg1, action);
    loginidlen = strlen(loginid);
    strcat(funcarg1, loginid, loginidlen);
    strcat(funcarg1, "\0");
    funcargs[1] = funcarg1;
    cidirlen = strlen(cidir);
    strncpy(funcarg2, cidir, cidirlen);
    strncat(funcarg2, "/s.", 3);

```

```

    streac(funcarg2, ci);
    streac(funcarg2, "\0");
    funcargs[2] = funcarg2;
    funcargs[3] = NULL;
    if (callscs(funcargs) != 0)
    {
        fprintf(stderr, "Update authorization not completed.\n");
        return(-1);
    }

    return(0);
}

/*****
/*  end authupdt                                     */
*****/

```

```

/*****
/*  callscs -- call an SCCS function
*****/

callscs(argv)

char *argv[];

    /* argv[0] = scs function to be executed */
    /* argv[1] - [last] = arguments to be passed to scs function */
    /* return = 0 if successful, -1 if unsuccessful */

    /* this routine does a fork system call to create a child */
    /* process, an execv system call to execute the requested scs */
    /* function, and waits in the parent process until the child */
    /* process finishes. */

{
    char funcpath[150];

    int cprocid;
    int wreturn;
    int wstatus;

    strncpy(funcpath, "/usr/bin/", 10);
    strcat(funcpath, argv[0]);

    /* create child process */
    cprocid = fork();
    if (cprocid == -1)
    {
        fprintf(stderr, "Fork failed for scsfunc(%s)\n", argv[0]);
        fprintf(stderr, "Errno = %d\n", errno);
        fprintf(stderr, "Argv = (%s)\n", argv);
        return(-1);
    }

    /* execute scs function in child */
    if (cprocid == 0)
    {
        if (execv(funcpath, argv) == -1)
        {
            fprintf(stderr, "Execv failed for scsfunc(%s)\n", argv[0]);
            fprintf(stderr, "Errno = %d\n", errno);
            fprintf(stderr, "Argv = (%s)\n", argv);
            return(-1);
        }
    }

    /* wait until child is done */
    wstatus = 0;

```

```

while ((wreturn = wait(&wstatus)) != cprocid);
if (wreturn == -1)
{
    fprintf(stderr, "SCCS function(%s) failed\n", argv[0]);
    fprintf(stderr, "Status = %d\n", wstatus);
    fprintf(stderr, "Errno = %d\n", errno);
    fprintf(stderr, "Return = %d\n", wreturn);
    fprintf(stderr, "Argv = (%s)\n", argv);
}

return(wstatus);
}

/*****
/* end callscs */
*****/

```

```

/*****
/* checkmr — check for valid Modification Request numbers */
*****/

checkmr(mrnumlist)

char mrnumlist[];

    /* mrnumlist = a list of MR numbers to be checked; numbers are */
    /*           separated by blanks and/or tabs; the list is */
    /*           terminated with an unescaped new-line character */
    /* return     = 0 if all numbers valid, -1 if any invalid */

{
    char mrnum[MAXVAL];

    int i;
    int rtnflag;
    int charent;

    rtnflag = 0;
    charent = 0;
    strncpy(mrnum, "\0", MAXVAL);

    /* get a character from mrnumlist and determine what it is */
    /* mrnumlist terminates with a new-line character */
    for (i = 0; mrnumlist[i] != '\n'; i++)
    {
        if (mrnumlist[i] == '\0')

            /* null */
            {
                continue;
            }
        else
        {
            if (mrnumlist[i] == ' ' || mrnumlist[i] == '\t')

                /* blank or tab */
                {
                    if (charent != 0)

                        /* no characters have been stored so this blank/tab */
                        /* terminates mrnum */

                        /* get mrnum */
                        {
                            mrnum[charent] = '\0';
                            if (getmr(mrnum, "g") == -1)
                            {
                                rtnflag = -1;

```

```

        }
        charcnt = 0;
        strncpy(mrnum, "\0", MAXVAL);
    }
}
else

/* store character */
{
    mrnum[charcnt] = mrnumlist[i];
    charcnt++;
}
}
}
if (charcnt != 0)

/* at end of list, and characters have been stored for */
/* the last mrnum */

/* get mrnum */
{
    charcnt++;
    mrnum[charcnt] = '\0';
    if (getmr(mrnum, "g") == -1)
    {
        rtnflag = -1;
    }
}
return(rtnflag);
}

/*****
/*  end checkmr */
*****/

```

```

/*****
/*  closemr — close a file associated with the          */
/*              Modification Request system              */
/*****

closemr(mrftp)

    /* mrftp  = filepointer to file to be closed          */
    /* return = 0 if successful, -1 if unsuccessful        */

{
    if ((fclose(mrftp)) == EOF)
    {
        fprintf(stderr, "Can't close file\n");
        return(-1);
    }
    else
    {
        return(0);
    }
}

/*****
/*  end closemr                                          */
/*****/

```

```

/*****
/*  getmr — get a specified Modification Request record      */
*****/

getmr(mrnum, getflag)

char mrnum[];
char getflag[];

    /* mrnum      = number of the Modification Request to be      */
    /*              retrieved                                         */
    /* getflag    = g to suppress actual retrieval of MR,         */
    /*              e if MR is to be retrieved and changed,        */
    /*              otherwise MR will be retrieved as read-only    */
    /* return     = 0 if successful, -1 if unsuccessful             */

{
    FILE *mrnumfp;
    FILE *hmrrecfp;

    char *funcargs[5];

    char funcarg0[200];
    char funcarg1[200];
    char funcarg2[200];
    char funcarg3[200];
    char mradminidir[100];
    char rmcmd[100];
    char chmodcmd[100];
    char mrfld[MAXFIELD];
    char hmrrec_file[150];

    int i;
    int j;
    int mrdirilen;
    int eofflag;

    /* get path for mradmin directory */
    if (pathmr(mradminidir) != 0)
    {
        fprintf(stdout, "Cannot get path for mradmin directory.\n");
        fprintf(stdout, "Please notify MR System Administrator.\n");
        return(-1);
    }

    /* initialize SCCS function argument areas */
    strncpy(funcarg0, "\0", 200);
    strncpy(funcarg1, "\0", 200);
    strncpy(funcarg2, "\0", 200);
    strncpy(funcarg3, "\0", 200);

```



```

/* call SCCS get function to access this MR */
strncpy(funcarg0, "get", 4);
funcargs[0] = funcarg0;
strncpy(funcarg1, "-s", 3);
funcargs[1] = funcarg1;
mrdirilen = strlen(mradmindir);
strcpy(funcarg2, mradmindir);
strncat(funcarg2, "/modreq/s.", 10);
strcat(funcarg2, mrnum);
funcargs[2] = funcarg2;
if (strcmp(getflag, "g") == 0)
{
    strncpy(funcarg3, "-g", 3);
    funcargs[3] = funcarg3;
}
else
{
    if (strcmp(getflag, "e") == 0)
    {
        strncpy(funcarg3, "-e", 3);
        funcargs[3] = funcarg3;
    }
    else
    {
        funcargs[3] = NULL;
    }
}
funcargs[4] = NULL;

if (callscs(funcargs) != 0)
{
    return(-1);
}

/* if suppressing retrieval, get out */
if (strcmp(getflag, "g") == 0)
    return(0);

/* SCCS get function puts MR record into a file in the */
/* user's current working directory */

/* initialize mrrec structure */
mrrecptr = mrrec;
for (i = 0; i < NUMKW; i++)
{
    strncpy(mrrecptr -> keyword, "\0", MAXKW);
    strncpy(mrrecptr -> value, "\0", MAXVAL);
    mrrecptr++;
}

/* open mr file in user's current working directory */

```

```

if (openmr(mrnum, "r") == -1)
{
    return(-1);
}
mrnumfp = mrfp;

/* copy MR record into mrrec structure */
mrrecptr = mrrec;
eofflag = 0;
while (eofflag == 0)
{
    if (fgets(mrflld, MAXFIELD, mrnumfp) != NULL)
    {
        if (strlen(mrflld) > 1)
        {
            i = 0;
            j = 0;
            while (mrflld[i] != ':')
            {
                mrrecptr -> keyword[j] = mrflld[i];
                i++;
                j++;
            }
            mrrecptr -> keyword[j] = '\0';
            i++;
            j = 0;
            while (mrflld[i] != '\n')
            {
                mrrecptr -> value[j] = mrflld[i];
                i++;
                j++;
            }
            mrrecptr -> value[j] = '\0';
            mrrecptr++;
        }
    }
    else
        eofflag = 1;
}

/* if MR is to be retrieved and updated, */
/* write structure to hold.mrrec file */
if (strcmp(getflag, "e") == 0)
{
    strcpy(hmrrec_file, mradmin_dir);
    strcat(hmrrec_file, "/hold.mrrec");
    if (openmr(hmrrec_file, "w") == -1)
        return(-1);
    hmrrecfp = mrfp;
    mrrecptr = mrrec;
    for (i = 0; i < NUMKW; i++)

```

```

    {
        fputs(mrrecptr -> keyword, hmrrecfp);
        putc(':', hmrrecfp);
        fputs(mrrecptr -> value, hmrrecfp);
        putc('\n', hmrrecfp);
        mrrecptr++;
    }
    if (closemr(hmrrecfp) == -1)
    {
        fprintf(stderr, "Can't close hold_mrrec_file.\n");
        return(-1);
    }
}

/* remove MR record from user's current working directory */
strcpy(chmodcmd, "chmod 666 ");
strcat(chmodcmd, mrnum);
if (system(chmodcmd) != 0)
{
    fprintf(stderr, "Change mode mrnum file in user dir. failed.\n");
    return(-1);
}
strcpy(rmcmd, "rm ");
strcat(rmcmd, mrnum);
if (system(rmcmd) != 0)
{
    fprintf(stderr, "Remove mrnum file from user directory failed.\n");
    return(-1);
}
return(0);
}

/*****
/* end getmr */
*****/

```

```

/*****
/*  listmr.c — list Modification Request contents  */
*****/

#include <stdio.h>
#include <strings.h>
#include <errno.h>
#include "mr.global.defs"
#include "openmr"
#include "closemr"
#include "getmr"
#include "callscs"
#include "pathmr"

main(argc,argv)

int argc;
char *argv[];

{
    FILE *hmrrecfp;
    FILE *listfp;

    char *cptr;
    char *lptr;

    char listtype[MAXINOPT];
    char mrnum[MAXVAL];
    char list_file[50];
    char stattype[MAXVAL];
    char contlist[MAXINOPT];

    int c;
    int i;
    int s;
    int eofmrnum;
    int validtype;
    int linecnt;

    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "*****\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "This routine lists Modification Requests.\n");

```

```

fprintf(stdout, "\n");

/* loop until no more lists requested */
for (;;)      /* this causes an infinite loop */
{
    /* get list type */
    validtype = 0;
    while(validtype == 0)
    {
        fprintf(stdout, "\n");
        fprintf(stdout, "Enter type of list as follows\n");
        fprintf(stdout, "      or\n");
        fprintf(stdout, "      carriage return to exit program.\n");
        fprintf(stdout, "\n");
        fprintf(stdout, "Enter:  To list:\n");
        fprintf(stdout, "-----\n");
        fprintf(stdout, "  a    all info. in MR\n");
        fprintf(stdout, "  c    all affected CIs listed in MR\n");
        fprintf(stdout, "  d    MR description\n");
        fprintf(stdout, "  s    associated info. for current status\n");
        fprintf(stdout, "\n");
        fprintf(stdout, "Enter type:\n");
        fgets(listtype, MAXINOPT, stdin);
        if ((cptr = index(listtype, '\n')) != 0)
            *cptr = NULL;
        if (strlen(listtype) == 0)
        {
            fprintf(stdout, "MR List program terminated by user.\n");
            exit(0);
        }

        /* check for valid list type entry */
        if ((strcmp(listtype, "a") == 0) ||
            (strcmp(listtype, "c") == 0) ||
            (strcmp(listtype, "d") == 0) ||
            (strcmp(listtype, "s") == 0))
            validtype = 1;
        else
            fprintf(stdout, "Invalid listtype %s\n", listtype);
    }

    /* get method of entry for mrnums */
    fprintf(stdout, "Do you want to enter the numbers of the MRs");
    fprintf(stdout, " to be listed\n");
    fprintf(stdout, " or read them in from a file?\n");
    fprintf(stdout, "Enter carriage return to enter MR numbers\n");
    fprintf(stdout, "      or\n");
    fprintf(stdout, "      name of file containing MR numbers.\n");
    fprintf(stdout, "\n");
    fgets(list_file, MAXINOPT, stdin);
    if ((cptr = index(list_file, '\n')) != 0)

```

```

    *cptr = NULL;

/* check if file name entered */
if (strlen(list_file) != 0)
{
    /* open file */
    if (openmr(list_file, "r") == -1)
    {
        fprintf(stdout, "MR sys. error -- can't open");
        fprintf(stdout, " %s file.\n", list_file);
        fprintf(stdout, "Please notify MR System Administrator.\n");
        exit(-1);
    }
}
listfp = mrfp;

/* list MRs until no more MR numbers entered */
eofmrnum = 0;
while (eofmrnum == 0)
{
    if (strlen(list_file) != 0)
    {
        /* get mrnum from file */
        if ((lptr = fgets(mrnum, MAXVAL, listfp)) == NULL)
        {
            fprintf(stderr, "%s file is empty.\n");
            eofmrnum = 1;
            continue;
        }
    }
    /* else
    {
        if (lptr == EOF)
        {
            eofmrnum = 1;
            continue;
        }
    } */
    if ((cptr = index(mrnum, '\n')) != 0)
        *cptr = NULL;
}
else
{
    /* get mrnum from terminal */
    fprintf(stdout, "\n");
    fprintf(stdout, "Enter number of MR to be listed\n");
    fprintf(stdout, "      or\n");
    fprintf(stdout, "      carriage return if no more MRs for");
    fprintf(stdout, " current list type.\n");
    fgets(mrnum, MAXINOPT, stdin);
    if ((cptr = index(mrnum, '\n')) != 0)
        *cptr = NULL;
}

```

```

    if (strlen(mrnum) == 0)
    {
        eofmrnum = 1;
        continue;
    }
}

/* get MR */
if (getmr(mrnum, " ") == 0)
{
    /* list requested info. */
    fprintf(stdout, "\n");
    c = listtype[0];
    switch(c)
    {
        case 'a':
            /* list all info. */
            linecnt = 0;
            mrrecptr = mrrec;
            for (i = 0; i < NUMKW; i++)
            {
                fprintf(stdout, "%s:", mrrecptr -> keyword);
                fprintf(stdout, "%s\n", mrrecptr -> value);
                mrrecptr++;
                linecnt++;
                if (linecnt == 20)
                {
                    fprintf(stdout, "Enter carriage return to continue");
                    fprintf(stdout, " listing fields in this MR\n");
                    fgets(contlist, MAXINOPT, stdin);
                    linecnt = 0;
                }
            }
            break;
        case 'c':
            /* list CIs */
            mrrecptr = mrrec;
            for (i = 0; i < NUMKW; i++)
            {
                if ((strcmp(mrrecptr -> keyword, "mrnum") == 0) ||
                    (strcmp(mrrecptr -> keyword, "ci1") == 0) ||
                    (strcmp(mrrecptr -> keyword, "ci2") == 0) ||
                    (strcmp(mrrecptr -> keyword, "ci3") == 0) ||
                    (strcmp(mrrecptr -> keyword, "ci4") == 0) ||
                    (strcmp(mrrecptr -> keyword, "ci5") == 0) ||
                    (strcmp(mrrecptr -> keyword, "ci6") == 0) ||
                    (strcmp(mrrecptr -> keyword, "ci7") == 0) ||
                    (strcmp(mrrecptr -> keyword, "ci8") == 0) ||
                    (strcmp(mrrecptr -> keyword, "ci9") == 0))
                {
                    fprintf(stdout, "%s:", mrrecptr -> keyword);

```

```

        fprintf(stdout, "%s\n", mrrecptr -> value);
    }
    mrrecptr++;
}
break;
case 'd':
    /* list desc */
    mrrecptr = mrrec;
    for (i = 0; i < NUMKW; i++)
    {
        if ((strcmp(mrrecptr -> keyword, "mrnum") == 0) ||
            (strcmp(mrrecptr -> keyword, "desc1") == 0) ||
            (strcmp(mrrecptr -> keyword, "desc2") == 0) ||
            (strcmp(mrrecptr -> keyword, "desc3") == 0) ||
            (strcmp(mrrecptr -> keyword, "desc4") == 0) ||
            (strcmp(mrrecptr -> keyword, "desc5") == 0))
        {
            fprintf(stdout, "%s:", mrrecptr -> keyword);
            fprintf(stdout, "%s\n", mrrecptr -> value);
        }
        mrrecptr++;
    }
    break;
case 's':
    /* list status info. */
    mrrecptr = mrrec;
    for (i = 0; i < NUMKW; i++)
    {
        if (strcmp(mrrecptr -> keyword, "mrnum") == 0)
        {
            fprintf(stdout, "%s:", mrrecptr -> keyword);
            fprintf(stdout, "%s\n", mrrecptr -> value);
        }
        if (strcmp(mrrecptr -> keyword, "status") == 0)
        {
            strcpy(stattype, mrrecptr -> value);
            fprintf(stdout, "%s:", mrrecptr -> keyword);
            fprintf(stdout, "%s\n", mrrecptr -> value);
        }
        mrrecptr++;
    }
    mrrecptr = mrrec;
    for (i = 0; i < NUMKW; i++)
    {
        /* list specific info. for each status */
        s = stattype[0];
        switch(s)
        {
            case 'i':
                /* assigned for investigation */
                if ((strcmp(mrrecptr -> keyword, "invid") == 0) ||

```



```

        (strcmp(mrrecptr -> keyword, "invdt") == 0))
    {
        fprintf(stdout, "%s:", mrrecptr -> keyword);
        fprintf(stdout, "%s\n", mrrecptr -> value);
    }
    break;
case 'a':
    /* accepted for implementation */
    if ((strcmp(mrrecptr -> keyword, "appid") == 0) ||
        (strcmp(mrrecptr -> keyword, "appdt") == 0))
    {
        fprintf(stdout, "%s:", mrrecptr -> keyword);
        fprintf(stdout, "%s\n", mrrecptr -> value);
    }
    break;
case 'r':
    /* rejected */
    if ((strcmp(mrrecptr -> keyword, "rejid") == 0) ||
        (strcmp(mrrecptr -> keyword, "rejdt") == 0) ||
        (strcmp(mrrecptr -> keyword, "rejreas1") == 0) ||
        (strcmp(mrrecptr -> keyword, "rejreas2") == 0) ||
        (strcmp(mrrecptr -> keyword, "rejreas3") == 0) ||
        (strcmp(mrrecptr -> keyword, "rejreas4") == 0) ||
        (strcmp(mrrecptr -> keyword, "rejreas5") == 0))
    {
        fprintf(stdout, "%s:", mrrecptr -> keyword);
        fprintf(stdout, "%s\n", mrrecptr -> value);
    }
    break;
case 'd':
    /* deferred */
    if ((strcmp(mrrecptr -> keyword, "defid") == 0) ||
        (strcmp(mrrecptr -> keyword, "defdt") == 0) ||
        (strcmp(mrrecptr -> keyword, "defreas1") == 0) ||
        (strcmp(mrrecptr -> keyword, "defreas2") == 0) ||
        (strcmp(mrrecptr -> keyword, "defreas3") == 0) ||
        (strcmp(mrrecptr -> keyword, "defreas4") == 0) ||
        (strcmp(mrrecptr -> keyword, "defreas5") == 0))
    {
        fprintf(stdout, "%s:", mrrecptr -> keyword);
        fprintf(stdout, "%s\n", mrrecptr -> value);
    }
    break;
case 'g':
    /* assigned for implementation */
    if ((strcmp(mrrecptr -> keyword, "asgnto1") == 0) ||
        (strcmp(mrrecptr -> keyword, "asgnto2") == 0) ||
        (strcmp(mrrecptr -> keyword, "asgnto3") == 0) ||
        (strcmp(mrrecptr -> keyword, "asgnto4") == 0) ||
        (strcmp(mrrecptr -> keyword, "asgnto5") == 0) ||
        (strcmp(mrrecptr -> keyword, "asgnto6") == 0) ||

```

```

        (strcmp(mrrecptr -> keyword, "asgnto7") == 0) ||
        (strcmp(mrrecptr -> keyword, "asgnto8") == 0) ||
        (strcmp(mrrecptr -> keyword, "asgnto9") == 0) ||
        (strcmp(mrrecptr -> keyword, "asgndt") == 0) ||
        (strcmp(mrrecptr -> keyword, "targdt") == 0))
    {
        fprintf(stdout, "%s:", mrrecptr -> keyword);
        fprintf(stdout, "%s\n", mrrecptr -> value);
    }
    break;
case 'm':
    /* completed */
    if ((strcmp(mrrecptr -> keyword, "compdt") == 0) ||
        (strcmp(mrrecptr -> keyword,
            "compdesc1") == 0) ||
        (strcmp(mrrecptr -> keyword,
            "compdesc2") == 0) ||
        (strcmp(mrrecptr -> keyword,
            "compdesc3") == 0) ||
        (strcmp(mrrecptr -> keyword,
            "compdesc4") == 0) ||
        (strcmp(mrrecptr -> keyword,
            "compdesc5") == 0))
    {
        fprintf(stdout, "%s:", mrrecptr -> keyword);
        fprintf(stdout, "%s\n", mrrecptr -> value);
    }
    break;
case 'c':
    /* closed */
    if ((strcmp(mrrecptr -> keyword, "closid") == 0) ||
        (strcmp(mrrecptr -> keyword, "closdt") == 0))
    {
        fprintf(stdout, "%s:", mrrecptr -> keyword);
        fprintf(stdout, "%s\n", mrrecptr -> value);
    }
    break;
    }
    mrrecptr++;
}
break;
}
}
else
    fprintf(stdout, "MR %s does not exist.\n", mrnum);
}
}
}

/*****/
/* end listmr.c */

```

/*****

```

/*****
/*  mr.global.defs -- MR System global definitions
*****/

FILE *mrfp;

#define MAXKW 10      /* maximum length of a keyword, including
/*                      terminating NULL
*/

#define MAXVAL 51     /* maximum length of a value, including
/*                      terminating NULL
*/

#define MAXFIELD 63   /* maximum length of one field in MR record
/*                      = MAXKW
/*                      + 1 (for ':')
/*                      + MAXVAL
/*                      + 1 (for newline)
*/

#define NUMKW 63      /* total number of valid keywords
*/

#define MAXINOPT 75   /* maximum length of a user input option,
/*                      (e.g., a status type entered to
/*                      select MRs for tracking, or an
/*                      option type to specify contents
/*                      of an MR to be listed
*/

/* this is the structure of a Modification Request record
/* a newline character is written at the end of each field
/* in the MR record
*/
struct mrfield
{
    char keyword[MAXKW];
    char colon;
    char value[MAXVAL];
};

struct mrfield mrrec[NUMKW];
struct mrfield *mrrecptr;

extern int errno;      /* required for system calls: fork,exec,wait
*/

/*****
/*  end mr.global.defs
*****/

```

```

: :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: :
:   mrsys -- Modification Request System Primary Shell                :
:   :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: :

```

```

option=0
echo ' '
echo ' '
echo 'Welcome to the Modification Request System.'
exec 2>>$HOME/../../mradmin/std.err

```

```

: if the mradmin directory is not located under the same directory    :
: as the login directories of the users, the previous exec statement  :
: must be commented out and the following five statements             :
: [echo,echo,echo,read,exec] must be made executable by removing the :
: comment characters                                                  :
: echo ' '                                                            :
: echo 'Please enter the full pathname for the mradmin directory.'      :
: echo 'For example, /sysid/projid/mradmin'                            :
: read mradmindir                                                    :
: exec 2>>$mradmindir/std.err                                          :

```

```

while true
do

```

```

    echo ' '
    echo ' '
    echo '*****'
    echo '*      Modification Request System Primary Function Menu      *'
    echo '*****'
    echo ' '
    echo 'The following primary functions are available:'
    echo ' '
    echo 'Option   Function'
    echo '-----'
    echo ' '
    echo '  0       Exit the Modification Request System'
    echo ' '
    echo '  1       Submit a Modification Request [MR Admin. only]'
    echo ' '
    echo '  2       Update a Modification Request [MR Admin. only]'
    echo ' '
    echo '  3       Select Modification Requests'
    echo ' '
    echo '  4       List Modification Requests'
    echo ' '
    echo '  5       Validate Modification Requests'
    echo ' '
    echo 'Enter option number for the function you wish to perform:'
    read option
    case $option in
        0) echo ' '
            echo 'Modification Request System terminated by user.'

```



```

/*****
/*  openmr — open a file associated with the          */
/*              Modification Request system            */
*****/

openmr(mrfile, iomode)

char mrfile[];
char iomode[];

    /* mrfile = name of file to be opened          */
    /* iomode = "r" if read, "w" if write,          */
    /* return = 0 if successful, -1 if unsuccessful  */

{
    if ((mrfp = fopen(mrfile, iomode)) == NULL)
    {
        fprintf(stderr, "Can't open file (%s) for %s\n", mrfile, iomode);
        return(-1);
    }
    else
    {
        return(0);
    }
}

/*****
/*  end openmr                                         */
*****/

```

```

/*****
/* pathci — setup path to configuration item directory */
*****/

pathci(cidir)

char cidir[100];

    /* cidir = full pathname for configuration item directory */
    /* return = 0 if successful, -1 if unsuccessful */

{
    char *cptr;

    char *getenv();    /* a function supplied with Standard UNIX C */
                      /* Library, which returns a char pointer */
                      /* to the value of an environment name */

    strncpy(cidir, "\0", 100);

    /* The following code sets up the default path, where the */
    /* CI Administrator's login directory (ciadmin) is under the */
    /* same directory as the login directories for the users of */
    /* the MR System and the configuration items are stored in a */
    /* directory (confitem) under the ciadmin directory. */
    /* If the directory structure is different, the following code */
    /* should be bypassed and the code following the comment */
    /* MODIFY HERE */
    /* should be used. */

    /* get path for CI directory in the project directory */
    /* for this user */
    cptr = getenv("HOME");
    strcpy(cidir, cptr);
    strncat(cidir, "../ciadmin/confitem", 20);
    return(0);

    /* MODIFY HERE if directory structure is different from default. */
    /* The user will have to enter the correct path. */
    /* The comment characters must be removed from the following */
    /* lines. */
    /* fprintf(stdout, "Please enter path for the configuration"); */
    /* fprintf(stdout, " item directory for your project.\n"); */
    /* fprintf(stdout, "The full pathname must be entered from"); */
    /* fprintf(stdout, " the root directory down to and"); */
    /* fprintf(stdout, " including the CI directory.\n"); */
    /* fprintf(stdout, "For example,"); */
    /* fprintf(stdout, " /sys.1/ci.data/projid.4/projfile.5\n"); */
    /* fgets(cidir, MAXINOPT, stdin); */

```



```
/* if ((cptr = index(cidir, '\n')) != 0) */
/*      *cptr = NULL; */
/* return(0); */
}

/*****
/* end pathci */
*****/
```

```

/*****
/* pathmr — setup path to mradmin directory */
*****/

pathmr(mradmindir)

char mradmindir[100];

    /* mradmindir = full pathname for mradmin directory */
    /* return = 0 if successful, -1 if unsuccessful */

{
    char *cptr;

    char *getenv();    /* a function supplied with Standard UNIX C */
                      /* Library, which returns a char pointer */
                      /* to the value of an environment name */

    strncpy(mradmindir, "\0", 100);

    /* The following code sets up the default path, where the */
    /* MR Administrator's login directory (mradmin) is under the */
    /* same directory as the login directories for the users of */
    /* that MR System. */
    /* If the directory structure is different, the following code */
    /* should be bypassed and the code following the comment */
    /* MODIFY HERE */
    /* should be used. */

    /* get path for mradmin directory in the project directory */
    /* for this user */
    cptr = getenv("HOME");
    strcpy(mradmindir, cptr);
    strncat(mradmindir, "../mradmin", 11);
    return(0);

    /* MODIFY HERE if directory structure is different from default. */
    /* The user will have to enter the correct path. */
    /* The comment characters must be removed from the following */
    /* lines. */
    /* fprintf(stdout, "Please enter path for the mradmin"); */
    /* fprintf(stdout, " directory for your project.\n"); */
    /* fprintf(stdout, "The full pathname must be entered from"); */
    /* fprintf(stdout, " the root directory down to and"); */
    /* fprintf(stdout, " including the mradmin directory.\n"); */
    /* fprintf(stdout, "For example, /sys.1/proj.a/mradmin\n"); */
    /* fgets(mradmindir, MAXNOPT, stdin); */
    /* if ((cptr = index(mradmindir, '\n')) != 0) */
    /* *cptr = NULL; */

```

```
    /* return(0); */
}

/*****
/* end pathmr */
*****/
```

```

/*****
/*  selectmr.c -- select all Modification Requests          */
/*                  that meet certain criteria              */
*****/

#include <stdio.h>
#include <strings.h>
#include <errno.h>
#include "mr.global.defs"
#include "openmr"
#include "closemr"
#include "getmr"
#include "callsccs"
#include "pathmr"

main(argc,argv)

int argc;
char *argv[];

{
    FILE *lmrnumfp;
    FILE *fmrnumfp;
    FILE *hmrrecfp;
    FILE *selfp;

    char *cptr;
    char *template;
    char *tempptr;

    char *mktemp();      /* a function, supplied with the Standard UNIX */
                        /* C Library, which makes a unique file name */

    char select[MAXINOPT];
    char mradmindir[100];
    char mrnum[MAXVAL];
    char lmrnum_file[150];
    char lmrnum[MAXVAL];
    char fmrnum_file[150];
    char fmrnum[MAXVAL];
    char dispflag[MAXINOPT];
    char select_file[50];
    char select_temp[50];

    int i;
    int mryear;
    int mrmonth;
    int mrseq;
    int mrdirlen;
    int optflag;
    int checkflag;

```

```

int lmrnumflag;
int selectcnt;

fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "*****\n");
fprintf(stdout, "\n");
fprintf(stdout, "This routine selects Modification Requests\n");
fprintf(stdout, "that meet certain criteria and stores their\n");
fprintf(stdout, "numbers in a file for later use.\n");
fprintf(stdout, "\n");
strcpy(select_temp, "mrnumXXXXXX");
template = select_temp;
temp_ptr = mktemp(template);
strcpy(select_file, temp_ptr);
fprintf(stdout, "The selected numbers will be stored in your\n");
fprintf(stdout, "current working directory in a file called ");
fprintf(stdout, " %s\n", select_file);
fprintf(stdout, "\n");
fprintf(stdout, "Do you also want to display the selected");
fprintf(stdout, " MR number(s)?\n");
fprintf(stdout, "Enter y or yes to display numbers\n");
fprintf(stdout, "      or\n");
fprintf(stdout, "      carriage return for no.\n");
fgets(displflag, MAXINOPT, stdin);
if ((cptr = index(displflag, '\n')) != 0)
    *cptr = NULL;

/* get path for mradmindir */
if (pathmr(mradmindir) != 0)
{
    fprintf(stdout, "Cannot get path for mradmindir.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}

/* get last MR number assigned */
strcpy(lmrnum_file, mradmindir);
strcat(lmrnum_file, "/last.mrnum");
if (openmr(lmrnum_file, "r") == -1)
{
    fprintf(stdout, "MR sys. error -- can't open last_mrnun_file.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}

```

```

}
lmrnumfp = mrfp;
if (fgets(lmrnum, MAXVAL, lmrnumfp) == NULL)
{
    fprintf(stderr, "Can't get from file (%s)\n", lmrnum_file);
    fprintf(stdout, "MR system error -- last_mrnum_file empty or");
    fprintf(stdout, " not open.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}
if ((cptr = index(lmrnum, '\n')) != 0)
    *cptr = NULL;

/* get first MR number assigned */
strcpy(fmrnum_file, mradmin_dir);
strcat(fmrnum_file, "/first_mrnum");
if (openmr(fmrnum_file, "r") == -1)
{
    fprintf(stdout, "MR sys. error -- can't open first_mrnum_file.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}
fmrnumfp = mrfp;
if (fgets(fmrnum, MAXVAL, fmrnumfp) == NULL)
{
    fprintf(stderr, "Can't get from file (%s)\n", fmrnum_file);
    fprintf(stdout, "MR system error -- first_mrnum_file empty or");
    fprintf(stdout, " not open.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}
if ((cptr = index(fmrnum, '\n')) != 0)
    *cptr = NULL;

/* loop to get input criteria and display selected MRs until */
/* carriage return is entered to exit program: */
for (;;) /* this causes an infinite loop */
{
    /* get selection criteria for MRs from standard input */
    optflag = 0;
    do
    {
        fprintf(stdout, "\n");
        fprintf(stdout, "Enter status type to be selected as follows\n");
        fprintf(stdout, "      or\n");
        fprintf(stdout, "      carriage return to exit program.\n");
        fprintf(stdout, "\n");
        fprintf(stdout, "Enter:   To select MRs with status of:   \n");
        fprintf(stdout, "-----\n");
        fprintf(stdout, "  a      approved for implementation   \n");
        fprintf(stdout, "  c      closed                         \n");
    }

```

```

fprintf(stdout, " d      deferred for later consideration\n");
fprintf(stdout, " g      assigned for implementation      \n");
fprintf(stdout, " i      assigned for investigation      \n");
fprintf(stdout, " m      implementation complete      \n");
fprintf(stdout, " n      new      \n");
fprintf(stdout, " r      rejected      \n");
fprintf(stdout, "\n");
fprintf(stdout, "Enter type:\n");
fgets(select, MAXINOPT, stdin);
if ((cptr = index(select, '\n')) != 0)
    *cptr = NULL;
if (strlen(select) == 0)
{
    fprintf(stdout, "MR Selection program terminated by user.\n");
    exit(0);
}

/* check for valid entry */
if ((strcmp(select, "a") == 0) ||
    (strcmp(select, "c") == 0) ||
    (strcmp(select, "d") == 0) ||
    (strcmp(select, "g") == 0) ||
    (strcmp(select, "i") == 0) ||
    (strcmp(select, "m") == 0) ||
    (strcmp(select, "n") == 0) ||
    (strcmp(select, "r") == 0))
{
    optflag = 1;
}
else
{
    fprintf(stdout, "Selection entry invalid.\n");
    fprintf(stdout, "Please re-enter.\n");
    fprintf(stdout, "\n");
}
} while (optflag = 0);
fprintf(stdout, "Selecting MRs -- please wait.\n");

/* open select file for write */
if (openmr(select_file, "w") == -1)
{
    fprintf(stdout, "MR sys. error -- can't open template file.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}
selfp = mrfp;

/* get MR with first_mrnum */
strcpy(mrnum, fmrnum);

/* loop through all MRs, checking and writing each */

```

```

/* MR number that contains requested data */
selectcnt = 0;
checkflag = 0;
fprintf(stdout, "\n");
while (checkflag == 0)
{
    if (getmr(mrnum, " ") == 0)
    {

        /* check to see if this MR meets the selection criteria */
        /* if so, put it out */
        mrrecptr = mrrec;
        for (i = 0; i < NUMKW; i++)
        {
            if (strcmp(mrrecptr->keyword, "status") == 0)
            {
                if (strcmp(mrrecptr->value, select) == 0)
                {
                    fputs(mrnum, selfp);
                    putc('\n', selfp);
                    if ((strcmp(disflag, "y") == 0) ||
                        (strcmp(disflag, "yes") == 0))
                        fprintf(stdout, "%s\n", mrnum);
                    selectcnt++;
                }
                break;
            }
            else
                mrrecptr++;
        }
    }
    if (strcmp(mrnum, lmrnum) != 0)
    {
        /* determine next mrnum */
        sscanf(mrnum, "%*2c%4d%2d%2d", &mryear, &mrmonth, &mrseq);
        if (mrseq == 99)
            if (mrmonth == 12)
            {
                mryear++;
                mrmonth = 1;
                mrseq = 1;
            }
            else
            {
                mrmonth++;
                mrseq=1;
            }
        else
            mrseq++;
        sprintf(mrnum, "mr%d%02d%02d", mryear, mrmonth, mrseq);
    }
}

```



```
    else
    {
        fprintf(stdout, "\n");
        fprintf(stdout, "%d MR numbers selected.\n", selectcnt);
        checkflag = 1;
    }
}
}

/*****
/*  end selectmr.c
*****/
```

```

/*****
/* submitmr.c — submit a Modification Request */
*****/

#include <stdio.h>
#include <errno.h>
#include <strings.h>
#include <time.h>
#include "mr.global.defs"
#include "openmr"
#include "closemr"
#include "callsecs"
#include "pathmr"

main(argc,argv)

int argc;
char *argv[];

{
    FILE *lmrnumfp;
    FILE *fmrnumfp;
    FILE *loginfp;
    FILE *hmrrecfp;

    char *cptr;

    char *cuserid();      /* a function supplied with Standard UNIX C */
                          /* Library, which returns a char pointer */
                          /* to the login name of the user */

    char *funcargs[7];

    char c[2];
    char mradmindir[100];
    char funcarg0[200];
    char funcarg1[200];
    char funcarg2[200];
    char funcarg3[200];
    char funcarg4[200];
    char funcarg5[200];
    char keynum[MAXKW];
    char mrnum[MAXVAL];
    char desc[5][MAXVAL];
    char origid[MAXVAL];
    char sysprojid[MAXVAL];
    char impact[5][MAXVAL];
    char prio[MAXVAL];
    char rcompdt[MAXVAL];
    char sever[MAXVAL];
    char lmrnum_file[150];

```

```

char lmrnum[MAXVAL];
char fmrnum_file[150];
char login_file[150];
char userid[L_cuserid];
char authid[15];
char hmrrec_file[150];

int i;
int curryear;
int currmo;
int mryear;
int mrmonth;
int mrseq;
int mrdirlen;
int authidlen;

long currttime;

/* initialize SCCS function argument areas */
strncpy(funcarg0, "\0", 200);
strncpy(funcarg1, "\0", 200);
strncpy(funcarg2, "\0", 200);
strncpy(funcarg3, "\0", 200);
strncpy(funcarg4, "\0", 200);
strncpy(funcarg5, "\0", 200);

fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "*****\n");
fprintf(stdout, "\n");
fprintf(stdout, "This routine submits Modification Requests.\n");
fprintf(stdout, "\n");

/* initialize standard input areas */
strncpy(mrnum, "\0", MAXVAL);
for (i = 0; i < 5; i++)
{
    strncpy(desc[i], "\0", MAXVAL);
}
strncpy(origid, "\0", MAXVAL);
strncpy(sysprojid, "\0", MAXVAL);
for (i = 0; i < 5; i++)
{
    strncpy(impact[i], "\0", MAXVAL);
}

```

```

}
strncpy(prio, "\0", MAXVAL);
strncpy(rcompdt, "\0", MAXVAL);
strncpy(sever, "\0", MAXVAL);

/* get path for mradmindir */
if (pathmr(mradmindir) != 0)
{
    fprintf(stdout, "Cannot get path for mradmindir.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}

/* get login id of person authorized to submit MRs */
strcpy(login_file, mradmindir);
strcat(login_file, "/auth.login");
if (openmr(login_file, "r") == -1)
{
    fprintf(stdout, "MR sys. error -- can't open auth_login_file.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}
loginfp = mrfp;
if (fgets(authid, 15, loginfp) == NULL)
{
    fprintf(stderr, "Can't get from file (%s)\n", login_file);
    fprintf(stdout, "MR system error -- auth_login_file empty or");
    fprintf(stdout, " not open.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}
if ((cptr = index(authid, '\n')) != 0)
    *cptr = NULL;

/* get login name of user */
cptr = userid;
if (cuserid(cptr) == NULL)
{
    fprintf(stdout, "Cannot find user's login name.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}

/* check that user is authorized to submit MRs */
if (strcmp(userid, authid) != 0)
{
    fprintf(stdout, "\n");
    fprintf(stdout, "You are not authorized to submit");
    fprintf(stdout, " Modification Requests.\n");
    fprintf(stdout, "This function is restricted to the");
    fprintf(stdout, " MR System Administrator.\n");
}

```

```

    exit(-1);
}

/* get last MR number assigned */
strcpy(lmrnum_file, mradmin_dir);
strcat(lmrnum_file, "/last.mrnum");
if (openmr(lmrnum_file, "r") == -1)
{
    fprintf(stdout, "MR sys. error -- can't open last_mrnun_file.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}
lmrnumfp = mrfp;
if (fgets(lmrnum, MAXVAL, lmrnumfp) == NULL)
{
    fprintf(stderr, "Can't get from file (%s)\n", lmrnum_file);
    fprintf(stdout, "MR system error -- last_mrnun_file empty or");
    fprintf(stdout, " not open.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}
if ((cptr = index(lmrnum, '\n')) != 0)
    *cptr = NULL;

/* get current date */
if (time(&currttime) == -1)
{
    fprintf(stderr, "Time call error -- errno (%d)\n", errno);
    fprintf(stdout, "MR system error -- can't get curr. time.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}
curryear = gmtime(&currttime) -> tm_year;
curryear = curryear + 1900;
currmo = gmtime(&currttime) -> tm_mon;
currmo++;

/* setup mrnum for this MR being submitted */
sscanf(lmrnum, "%*2c%4d%2d%2d", &mryear, &mrmonth, &mrseq);
if (mryear == curryear)
    if (mrmonth == currmo)
        if (mrseq == 99)
        {
            fprintf(stderr, "mrseq at upper limit\n");
            fprintf(stdout, "Can't create > 99 MRs/month.\n");
            fprintf(stdout, "Please notify MR System administrator.\n");
            exit(-1);
        }
        else
            mrseq++;
    else

```

```

    {
        mrmonth = currmo;
        mrseq = 1;
    }
else
{
    mryear = curryear;
    mrmonth = currmo;
    mrseq = 1;
}
sprintf(mrnum, "mr%d%02d%02d", mryear, mrmonth, mrseq);

/* get input values for MR fields from standard input */
fprintf(stdout, "The following information is required.\n");
fprintf(stdout, "Please keep entry to max. of 50 characters:\n");
fprintf(stdout, "\n");
fprintf(stdout, "Enter description of modification requested.\n");
fprintf(stdout, "Up to 5 lines of description can be entered.\n");
fprintf(stdout, "Enter carriage return if skipping a line:\n");
fprintf(stdout, ".....1.....2.....3.....4");
fprintf(stdout, ".....5\n");
fprintf(stdout, "Description line 1:\n");
fgets(desc[0], MAXVAL, stdin);
if ((cptr = index(desc[0], '\n')) != 0)
    *cptr = NULL;
while (strlen(desc[0]) == 0)
{
    fprintf(stdout, "Please enter description of modification");
    fprintf(stdout, " requested.\n");
    fprintf(stdout, ".....1.....2.....3.....4");
    fprintf(stdout, ".....5\n");
    fprintf(stdout, "Description line 1:\n");
    fgets(desc[0], MAXVAL, stdin);
    if ((cptr = index(desc[0], '\n')) != 0)
        *cptr = NULL;
}
fprintf(stdout, "Description line 2:\n");
fgets(desc[1], MAXVAL, stdin);
if ((cptr = index(desc[1], '\n')) != 0)
    *cptr = NULL;
fprintf(stdout, "Description line 3:\n");
fgets(desc[2], MAXVAL, stdin);
if ((cptr = index(desc[2], '\n')) != 0)
    *cptr = NULL;
fprintf(stdout, "Description line 4:\n");
fgets(desc[3], MAXVAL, stdin);
if ((cptr = index(desc[3], '\n')) != 0)
    *cptr = NULL;
fprintf(stdout, "Description line 5:\n");
fgets(desc[4], MAXVAL, stdin);
if ((cptr = index(desc[4], '\n')) != 0)

```

```

    *cptr = NULL;
    fprintf(stdout, "Enter originator id.\n");
    fprintf(stdout, ".....1.....2.....3.....4");
    fprintf(stdout, ".....5\n");
    fgets(origid, MAXVAL, stdin);
    if ((cptr = index(origid, '\n')) != 0)
        *cptr = NULL;
    while (strlen(origid) == 0)
    {
        fprintf(stdout, "Please enter originator id.\n");
        fprintf(stdout, ".....1.....2.....3.....4");
        fprintf(stdout, ".....5\n");
        fgets(origid, MAXVAL, stdin);
        if ((cptr = index(origid, '\n')) != 0)
            *cptr = NULL;
    }
    fprintf(stdout, "Enter system/project id.\n");
    fprintf(stdout, ".....1.....2.....3.....4");
    fprintf(stdout, ".....5\n");
    fgets(sysprojid, MAXVAL, stdin);
    if ((cptr = index(sysprojid, '\n')) != 0)
        *cptr = NULL;
    while (strlen(sysprojid) == 0)
    {
        fprintf(stdout, "Please enter system/project id.\n");
        fprintf(stdout, ".....1.....2.....3.....4");
        fprintf(stdout, ".....5\n");
        fgets(sysprojid, MAXVAL, stdin);
        if ((cptr = index(sysprojid, '\n')) != 0)
            *cptr = NULL;
    }
    fprintf(stdout, "\n");
    fprintf(stdout, "The following information is optional.\n");
    fprintf(stdout, "Please keep entry to max. of 50 characters.\n");
    fprintf(stdout, "Enter carriage return if skipping entry:\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "Enter impact of modification requested.\n");
    fprintf(stdout, "Up to 5 lines of impact can be entered.\n");
    fprintf(stdout, "Enter carriage return if skipping a line:\n");
    fprintf(stdout, ".....1.....2.....3.....4");
    fprintf(stdout, ".....5\n");
    fprintf(stdout, "Impact line 1:\n");
    fgets(impact[0], MAXVAL, stdin);
    if ((cptr = index(impact[0], '\n')) != 0)
        *cptr = NULL;
    fprintf(stdout, "Impact line 2:\n");
    fgets(impact[1], MAXVAL, stdin);
    if ((cptr = index(impact[1], '\n')) != 0)
        *cptr = NULL;
    fprintf(stdout, "Impact line 3:\n");
    fgets(impact[2], MAXVAL, stdin);

```

```

if ((cptr = index(impact[2], '\n')) != 0)
    *cptr = NULL;
fprintf(stdout, "Impact line 4:\n");
fgets(impact[3], MAXVAL, stdin);
if ((cptr = index(impact[3], '\n')) != 0)
    *cptr = NULL;
fprintf(stdout, "Impact line 5:\n");
fgets(impact[4], MAXVAL, stdin);
if ((cptr = index(impact[4], '\n')) != 0)
    *cptr = NULL;
fprintf(stdout, "Enter priority of request.\n");
fprintf(stdout, ".....1.....2.....3.....4");
fprintf(stdout, ".....5\n");
fgets(prio, MAXVAL, stdin);
if ((cptr = index(prio, '\n')) != 0)
    *cptr = NULL;
fprintf(stdout, "Enter requested completion date of");
fprintf(stdout, " modification.\n");
fprintf(stdout, ".....1.....2.....3.....4");
fprintf(stdout, ".....5\n");
fgets(rcompdt, MAXVAL, stdin);
if ((cptr = index(rcompdt, '\n')) != 0)
    *cptr = NULL;
fprintf(stdout, "Enter severity of modification requested.\n");
fprintf(stdout, ".....1.....2.....3.....4");
fprintf(stdout, ".....5\n");
fgets(sever, MAXVAL, stdin);
if ((cptr = index(sever, '\n')) != 0)
    *cptr = NULL;
fprintf(stdout, "Creating MR — please wait.\n");

/* write keywords and input values to mrrec structure */
mrrecptr = mrrec;
strncpy(mrrecptr->keyword, "mrnum", MAXKW);
strncpy(mrrecptr->value, mrnum, MAXVAL);
mrrecptr++;
for (i = 0; i < 5; i++)
{
    strncpy(keynum, "\0", MAXKW);
    strncpy(keynum, "desc", 4);
    c[0] = i + 1 + '0';
    c[1] = '\0';
    strcat(keynum, c);
    strncpy(mrrecptr->keyword, keynum, MAXKW);
    strncpy(mrrecptr->value, desc[i], MAXVAL);
    mrrecptr++;
}
strncpy(mrrecptr->keyword, "origid", MAXKW);
strncpy(mrrecptr->value, origid, MAXVAL);
mrrecptr++;
strncpy(mrrecptr->keyword, "sysprojid", MAXKW);

```



```

strncpy(mrrecptr -> value, sysprojid, MAXVAL);
mrrecptr++;
for (i = 0; i < 5; i++)
{
    strncpy(keynum, "\0", MAXKW);
    strncpy(keynum, "impact", 6);
    c[0] = i + 1 + '0';
    c[1] = '\0';
    strcat(keynum, c);
    strncpy(mrrecptr -> keyword, keynum, MAXKW);
    strncpy(mrrecptr -> value, impact[i], MAXVAL);
    mrrecptr++;
}
strncpy(mrrecptr -> keyword, "prio", MAXKW);
strncpy(mrrecptr -> value, prio, MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "rcompdt", MAXKW);
strncpy(mrrecptr -> value, rcompdt, MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "sever", MAXKW);
strncpy(mrrecptr -> value, sever, MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "status", MAXKW);
strncpy(mrrecptr -> value, "n", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "invid", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "invdt", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "appid", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "appdt", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "rejid", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "rejdt", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
for (i = 0; i < 5; i++)
{
    strncpy(keynum, "\0", MAXKW);
    strncpy(keynum, "rejas", 7);
    c[0] = i + 1 + '0';
    c[1] = '\0';
    strcat(keynum, c);
    strncpy(mrrecptr -> keyword, keynum, MAXKW);

```

```

    strncpy(mrrecptr -> value, "\0", MAXVAL);
    mrrecptr++;
}
strncpy(mrrecptr -> keyword, "defid", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "defdt", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
for (i = 0; i < 5; i++)
{
    strncpy(keynum, "\0", MAXKW);
    strncpy(keynum, "defreas", 7);
    c[0] = i + 1 + '0';
    c[1] = '\0';
    strcat(keynum, c);
    strncpy(mrrecptr -> keyword, keynum, MAXKW);
    strncpy(mrrecptr -> value, "\0", MAXVAL);
    mrrecptr++;
}
strncpy(mrrecptr -> keyword, "asgnto1", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "asgnto2", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "asgnto3", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "asgnto4", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "asgnto5", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "asgnto6", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "asgnto7", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "asgnto8", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "asgnto9", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "asgndt", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "targdt", MAXKW);

```

```

strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "compdt", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
for (i = 0; i < 5; i++)
{
    strncpy(keynum, "\0", MAXKW);
    strncpy(keynum, "compdesc", 8);
    c[0] = i + 1 + '0';
    c[1] = '\0';
    strcat(keynum, c);
    strncpy(mrrecptr -> keyword, keynum, MAXKW);
    strncpy(mrrecptr -> value, "\0", MAXVAL);
    mrrecptr++;
}
for (i = 0; i < 9; i++)
{
    strncpy(keynum, "\0", MAXKW);
    strncpy(keynum, "ci", 2);
    c[0] = i + 1 + '0';
    c[1] = '\0';
    strcat(keynum, c);
    strncpy(mrrecptr -> keyword, keynum, MAXKW);
    strncpy(mrrecptr -> value, "\0", MAXVAL);
    mrrecptr++;
}
strncpy(mrrecptr -> keyword, "closid", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);
mrrecptr++;
strncpy(mrrecptr -> keyword, "closdt", MAXKW);
strncpy(mrrecptr -> value, "\0", MAXVAL);

/* write structure to hold.mrrec file */
strcpy(hmrrec_file, mradmindir);
strcat(hmrrec_file, "/hold.mrrec");
if (openmr(hmrrec_file, "w") == -1)
{
    fprintf(stdout, "MR sys. error -- can't open hold_mrrec_file.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}
hmrrecfp = mrfp;
mrrecptr = mrrec;
for (i = 0; i < NUMKW; i++)
{
    fputs(mrrecptr -> keyword, hmrrecfp);
    putc(':', hmrrecfp);
    fputs(mrrecptr -> value, hmrrecfp);
    putc('\n', hmrrecfp);
    mrrecptr++;
}

```

```

}
if (closemr(hmrrecfp) == -1)
{
    fprintf(stdout, "MR sys. error -- can't close hold_mrrec_file.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}

/* call SCCS admin function to create this MR as an SCCS file */
strncpy(funcarg0, "admin", 6);
funcargs[0] = funcarg0;
strncpy(funcarg1, "-n", 3);
funcargs[1] = funcarg1;
strncpy(funcarg2, "-i", 2);
strcat(funcarg2, mradminidir);
strcat(funcarg2, "/hold.mrrec");
funcargs[2] = funcarg2;
strncpy(funcarg3, "-fqModification-Request", 24);
funcargs[3] = funcarg3;
strncpy(funcarg4, "-a", 2);
authidlen = strlen(authid);
strncat(funcarg4, authid, authidlen);
strcat(funcarg4, "\0");
funcargs[4] = funcarg4;
mrdirilen = strlen(mradminidir);
strncpy(funcarg5, mradminidir, mrdirilen);
strncat(funcarg5, "/modreq/s.", 10);
strcat(funcarg5, mrnum);
funcargs[5] = funcarg5;
funcargs[6] = NULL;
if (callscs(funcargs) != 0)
{
    fprintf(stdout, "Modification Request not created.\n");
    exit(-1);
}

/* update file containing last MR number assigned */
if (openmr(lmrnum_file, "w") == -1)
{
    fprintf(stdout, "MR sys. error --");
    fprintf(stdout, " can't open last_mrnum_file.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}
lmrnumfp = mrfp;
fputs(mrnum, lmrnumfp);
putc('\n', lmrnumfp);
if (closemr(lmrnumfp) == -1)
{
    fprintf(stdout, "MR sys. error -- can't close last_mrnum_file.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
}

```

```

    exit(-1);
}

/* check to see if this was first MR created in mradminidir */
/* if so, store mrnum in file containing first MR number assigned */
if (strcmp(lmrnum, "0000000000", 10) == 0)
{
    strcpy(fmrnum_file, mradminidir);
    strcat(fmrnum_file, "/first.mrnum");
    if (openmr(fmrnum_file, "w") == -1)
    {
        fprintf(stdout, "MR sys. error --");
        fprintf(stdout, " can't open first_mrnum_file.\n");
        fprintf(stdout, "Please notify MR System Administrator.\n");
        exit(-1);
    }
    fmrnumfp = mrfp;
    fputs(mrnum, fmrnumfp);
    puts("\n", fmrnumfp);
    if (closemr(fmrnumfp) == -1)
    {
        fprintf(stdout, "MR system error --");
        fprintf(stdout, " can't close first_mrnum_file.\n");
        fprintf(stdout, "Please notify MR System Administrator.\n");
        exit(-1);
    }
}

fprintf(stdout, "\n");
fprintf(stdout, "Submission complete.\n");
fprintf(stdout, "\n");
fprintf(stdout, "Modification Request number  %s", mrnum);
fprintf(stdout, " created.\n");
fprintf(stdout, "\n");
fprintf(stdout, "*****\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");

exit(0);
}

/*****
/* end submitmr.c */
*****/

```

```

/*****
/*  updatemr.c — update Modification Requests      */
*****/

#include <stdio.h>
#include <strings.h>
#include <errno.h>
#include <time.h>
#include "mr.global.defs"
#include "authupdt"
#include "openmr"
#include "closemr"
#include "getmr"
#include "callscs"
#include "pathci"
#include "pathmr"

main(argc,argv)

int argc;
char *argv[];

{
    FILE *hmrrecfp;
    FILE *loginfp;

    char *cptr;

    char *cuserid();      /* a function supplied with Standard UNIX C */
                        /* Library, which returns a char pointer */
                        /* to the login name of the user          */

    char *funcargs[5];

    char charj[2];
    char mradmindir[100];
    char mrnum[MAXVAL];
    char fieldid[MAXINOPT];
    char dispflag[MAXINOPT];
    char funcarg0[200];
    char funcarg1[200];
    char funcarg2[200];
    char funcarg3[200];
    char hmrrec_file[150];
    char login_file[150];
    char rmcmd[100];
    char cpcmd[100];
    char loginid[MAXINOPT];
    char currdte[9];
    char newval[MAXVAL];
    char ciid[MAXINOPT];
    char userid[L_cuserid];

```

```

char authid[15];
char asgnto[8];
char ci[4];
char asgntoid[9][MAXVAL];

int i;
int j;
int k;
int endupdate;
int nodisplay;
int abort;
int curryear;
int currmo;
int currday;
int mrdirlen;
int fldidint;
int keywdid;
int idend;
int idstored;

long currttime;

static char keywdin[NUMKW][MAXKW] = {
    "desc1",
    "desc2",
    "desc3",
    "desc4",
    "desc5",
    "origid",
    "sysprojid",
    "impact1",
    "impact2",
    "impact3",
    "impact4",
    "impact5",
    "prio",
    "rcompdt",
    "sever",
    "status",
    "invid",
    "appid",
    "rejid",
    "rejreas1",
    "rejreas2",
    "rejreas3",
    "rejreas4",
    "rejreas5",
    "defid",
    "defreas1",
    "defreas2",
    "defreas3",

```

```

    "defreas4",
    "defreas5",
    "asgnto1",
    "asgnto2",
    "asgnto3",
    "asgnto4",
    "asgnto5",
    "asgnto6",
    "asgnto7",
    "asgnto8",
    "asgnto9",
    "targdt",
    "compdesc1",
    "compdesc2",
    "compdesc3",
    "compdesc4",
    "compdesc5",
    "ci1",
    "ci2",
    "ci3",
    "ci4",
    "ci5",
    "ci6",
    "ci7",
    "ci8",
    "ci9",
    "closid"
};

/* initialize SCCS function argument areas */
strncpy(funcarg0, "\0", 200);
strncpy(funcarg1, "\0", 200);
strncpy(funcarg2, "\0", 200);
strncpy(funcarg3, "\0", 200);

fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "\n");
fprintf(stdout, "*****\n");
fprintf(stdout, "\n");
fprintf(stdout, "This routine updates Modification Requests.\n");
fprintf(stdout, "\n");

/* get path for mradmin directory */
if (pathmr(mradmin) != 0)

```



```

{
    fprintf(stdout, "Cannot get path for mradmin directory.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}

/* get login id of person authorized to submit MRs */
strcpy(login_file, mradmindir);
strcat(login_file, "/auth.login");
if (openmr(login_file, "r") == -1)
{
    fprintf(stdout, "MR sys. error -- can't open auth_login_file.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}
loginfp = mrfp;
if (fgets(authid, 15, loginfp) == NULL)
{
    fprintf(stderr, "Can't get from file (%s)\n", login_file);
    fprintf(stdout, "MR system error -- auth_login_file empty or");
    fprintf(stdout, " not open.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}
if ((cptr = index(authid, '\n')) != 0)
    *cptr = NULL;

/* get login name of user */
cptr = userid;
if (cuserid(cptr) == NULL)
{
    fprintf(stdout, "Cannot find user's login name.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}

/* check that user is authorized to submit MRs */
if (strcmp(userid, authid) != 0)
{
    fprintf(stdout, "\n");
    fprintf(stdout, "You are not authorized to submit");
    fprintf(stdout, " Modification Requests.\n");
    fprintf(stdout, "This function is restricted to the");
    fprintf(stdout, " MR System Administrator.\n");
    exit(-1);
}

/* get current date */
if (time(&currttime) == -1)
{
    fprintf(stderr, "Time call error -- errno (%d)\n", errno);
}

```

```

    fprintf(stdout, "MR system error -- can't get curr. time.\n");
    fprintf(stdout, "Please notify MR System Administrator.\n");
    exit(-1);
}
curryear = gmtime(&currtime) -> tm_year;
curryear = curryear + 1900;
currmo   = gmtime(&currtime) -> tm_mon;
currmo++;
currday  = gmtime(&currtime) -> tm_mday;
sprintf(currdate, "%4d%02d%02d", curryear, currmo, currday);
currdate[8] = '\0';

/* loop until no more MRs to be updated */
for (;;)          /* this causes an infinite loop */
{
    fprintf(stdout, "Enter number of MR to be updated,\n");
    fprintf(stdout, "          or\n");
    fprintf(stdout, "          carriage return to exit.\n");
    fgets(mrnum, MAXVAL, stdin);
    if ((cptr = index(mrnum, '\n')) != 0)
        *cptr = NULL;
    if (strlen(mrnum) == 0)

        /* no input so terminate program */
        {
            fprintf(stdout, "MR Update program terminated by user.\n");
            exit(0);
        }

    /* get MR */
    if (getmr(mrnum, "e") == 0)
    {
        /* loop until no more fields to be changed in this MR */
        endupdate = 0;
        while (endupdate == 0)
        {
            /* loop to get id of field to be changed */
            nodisplay = 0;
            while (nodisplay == 0)
            {
                fprintf(stdout, "\n");
                fprintf(stdout, "Enter id number of field to be changed\n");
                fprintf(stdout, "          (if doing multiple changes to");
                fprintf(stdout, " this MR and status code is one of");
                fprintf(stdout, " them,\n");
                fprintf(stdout, "          status code should be changed");
                fprintf(stdout, " first\n");
                fprintf(stdout, "          or other changes might not be");
                fprintf(stdout, " accepted), or\n");
                fprintf(stdout, "          carriage return to display list of");
                fprintf(stdout, " field id numbers, or\n");
            }
        }
    }
}

```

```

fprintf(stdout, "      end or END to abort update of");
fprintf(stdout, " this MR, or\n");
fprintf(stdout, "      0 to save updated MR.\n");
fgets(fieldid, MAXINOPT, stdin);
if ((cptr = index(fieldid, '\n')) != 0)
    *cptr = NULL;
if (strlen(fieldid) == 0)
{
    /* display list of field ids */
    fprintf(stdout, "\n");
    fprintf(stdout, "Id  Field Name  ");
    fprintf(stdout, "  Id  Field Name  ");
    fprintf(stdout, "  Id  Field Name  ");
    fprintf(stdout, "  Id  Field Name\n");
    fprintf(stdout, "--  -----");
    fprintf(stdout, "  --  -----");
    fprintf(stdout, "  --  -----");
    fprintf(stdout, "  --  -----");
    fprintf(stdout, " 1  Description 1  ");
    fprintf(stdout, " 16  Status      ");
    fprintf(stdout, " 31  Assign to id 1");
    fprintf(stdout, " 46  Config.item 1\n");
    fprintf(stdout, " 2  Description 2  ");
    fprintf(stdout, " 17  Investig. id ");
    fprintf(stdout, " 32  Assign to id 2");
    fprintf(stdout, " 47  Config.item 2\n");
    fprintf(stdout, " 3  Description 3  ");
    fprintf(stdout, " 18  Approved id  ");
    fprintf(stdout, " 33  Assign to id 3");
    fprintf(stdout, " 48  Config.item 3\n");
    fprintf(stdout, " 4  Description 4  ");
    fprintf(stdout, " 19  Rejected id  ");
    fprintf(stdout, " 34  Assign to id 4");
    fprintf(stdout, " 49  Config.item 4\n");
    fprintf(stdout, " 5  Description 5  ");
    fprintf(stdout, " 20  Reject reas.1");
    fprintf(stdout, " 35  Assign to id 5");
    fprintf(stdout, " 50  Config.item 5\n");
    fprintf(stdout, " 6  Originator id  ");
    fprintf(stdout, " 21  Reject reas.2");
    fprintf(stdout, " 36  Assign to id 6");
    fprintf(stdout, " 51  Config.item 6\n");
    fprintf(stdout, " 7  System/proj. id");
    fprintf(stdout, " 22  Reject reas.3");
    fprintf(stdout, " 37  Assign to id 7");
    fprintf(stdout, " 52  Config.item 7\n");
    fprintf(stdout, " 8  Impact 1      ");
    fprintf(stdout, " 23  Reject reas.4");
    fprintf(stdout, " 38  Assign to id 8");
    fprintf(stdout, " 53  Config.item 8\n");
    fprintf(stdout, " 9  Impact 2      ");

```

```

        fprintf(stdout, "    24 Reject reas.5");
        fprintf(stdout, "    39 Assign to id 9");
        fprintf(stdout, "    54 Config.item 9\n");
        fprintf(stdout, "10 Impact 3      ");
        fprintf(stdout, "    25 Deferred id  ");
        fprintf(stdout, "    40 Target date  ");
        fprintf(stdout, "    55 Closed id\n");
        fprintf(stdout, "11 Impact 4      ");
        fprintf(stdout, "    26 Defer.reas. 1");
        fprintf(stdout, "    41 Compl. desc. 1\n");
        fprintf(stdout, "12 Impact 5      ");
        fprintf(stdout, "    27 Defer.reas. 2");
        fprintf(stdout, "    42 Compl. desc. 2\n");
        fprintf(stdout, "13 Priority      ");
        fprintf(stdout, "    28 Defer.reas. 3");
        fprintf(stdout, "    43 Compl. desc. 3\n");
        fprintf(stdout, "14 Request.comp.dt");
        fprintf(stdout, "    29 Defer.reas. 4");
        fprintf(stdout, "    44 Compl. desc. 4\n");
        fprintf(stdout, "15 Severity      ");
        fprintf(stdout, "    30 Defer.reas. 5");
        fprintf(stdout, "    45 Compl. desc. 5\n");
        continue;
    }

    if ((strcmp(fieldid, "end") == 0) ||
        (strcmp(fieldid, "END") == 0))
        /* abort update of MR */
    {
        strcpy(rmcmd, "rm ");
        strcat(rmcmd, mradminidir);
        strncat(rmcmd, "/modreq/p.", 10);
        strcat(rmcmd, mrnum);
        if (system(rmcmd) != 0)
        {
            fprintf(stderr, "Remove p-file failed.\n");
            fprintf(stdout, "MR System error during abort.\n");
            fprintf(stdout, "Please notify MR System");
            fprintf(stdout, " Administrator.\n");
            exit(-1);
        }
        endupdate = 1;
        fprintf(stdout, "\n");
        fprintf(stdout, "Update of MR");
        fprintf(stdout, " %s aborted by user.\n", mrnum);
        fprintf(stdout, "\n");
        break;
    }

    if (strcmp(fieldid, "0") == 0)
        /* write structure to hold.mrrec file */

```

```

{
    fprintf(stdout, "Saving MR — please wait.\n");
    strcpy(hmrrec_file, mradmindir);
    strcat(hmrrec_file, "/hold.mrrec");
    if (openmr(hmrrec_file, "w") == -1)
    {
        fprintf(stdout, "Please notify MR System");
        fprintf(stdout, " Administrator.\n");
        exit(-1);
    }
    hmrrecfp = mrfp;
    mrrecptr = mrrec;
    for (i = 0; i < NUMKW; i++)
    {
        fputs(mrrecptr -> keyword, hmrrecfp);
        putc(':', hmrrecfp);
        fputs(mrrecptr -> value, hmrrecfp);
        putc('\n', hmrrecfp);
        mrrecptr++;
    }
    if (closemr(hmrrecfp) == -1)
    {
        fprintf(stdout, "MR sys. error --");
        fprintf(stdout, "can't close hold_mrrec_file.\n");
        fprintf(stdout, "Please notify MR System");
        fprintf(stdout, " Administrator.\n");
        exit(-1);
    }

    /* copy hold.mrrec to file named mrnum as required */
    /* for delta */
    strcpy(cpcmd, "cp ");
    strcat(cpcmd, mradmindir);
    strncat(cpcmd, "/hold.mrrec ", 12);
    strcat(cpcmd, mrnum);
    if (system(cpcmd) != 0)
    {
        fprintf(stderr, "Copy hold.mrrec to mrnum failed.\n");
        fprintf(stdout, "Please notify MR System");
        fprintf(stdout, " Administrator.\n");
        exit(-1);
    }

    /* call SCCS delta function to write a delta to this */
    /* MR SCCS file */
    strncpy(funcarg0, "delta", 6);
    funcargs[0] = funcarg0;
    strncpy(funcarg1, "-s", 3);
    funcargs[1] = funcarg1;
    strncpy(funcarg2, "-y[]", 5);
    funcargs[2] = funcarg2;

```

```

mrdirilen = strlen(mradmindir);
strncpy(funcarg3, mradmindir, mrdirilen);
strncat(funcarg3, "/modreq/s.", 10);
strcat(funcarg3, mrnum);
funcargs[3] = funcarg3;
funcargs[4] = NULL;
if (callsccs(funcargs) != 0)
{
    fprintf(stdout, "MR System error --");
    fprintf(stdout, " Modification Request not updated.\n");
    fprintf(stdout, "Please notify MR System");
    fprintf(stdout, " Administrator.\n");
    exit(-1);
}
endupdate = 1;
fprintf(stdout, "\n");
fprintf(stdout, "MR Updated\n");
fprintf(stdout, "\n");
break;
}

/* field id was entered */
/* convert fieldid from string to integer */
fldidint = atoi(fieldid);

/* check for valid field id */
if ((fldidint >= 1) && (fldidint <= 55))
{
    keywddid = fldidint - 1;
    nodisplay = 1;
}
else
    fprintf(stdout, "Invalid fieldid %d\n", fldidint);
}

if (endupdate == 1)
    break;

/* get new value */
fprintf(stdout, "Enter new value for %s,\n", keywdin[keywddid]);
fprintf(stdout, "      or\n");
fprintf(stdout, "      DELETE to delete current value,\n");
fprintf(stdout, "      or\n");
fprintf(stdout, "      carriage return to enter a");
fprintf(stdout, " new field id.\n");
fprintf(stdout, "Please keep entry to max. of 50 chars.\n");
fprintf(stdout, ".....1.....2.....3.....4");
fprintf(stdout, ".....5\n");
fgets(newval, MAXVAL, stdin);
if ((cptr = index(newval, '\n')) != 0)
    *cptr = NULL;

```

```

/* check if user wants to enter new field id */
if (strlen(newval) == 0)
    continue;

/* check that required field is not being deleted */
if (strcmp(newval, "DELETE") == 0)
{
    if ((fldidint == 1) ||      /* desc1      */
        (fldidint == 6) ||      /* origid     */
        (fldidint == 7) ||      /* sysprojid  */
        (fldidint == 16))      /* status     */
    {
        fprintf(stdout, "%s is required field", keywdin[keywddid]);
        fprintf(stdout, " and cannot be deleted.\n");
        continue;
    }
}

/* check that status is not being changed to new */
if ((fldidint == 16) && (strcmp(newval, "n") == 0))
{
    fprintf(stdout, "Status cannot be changed to 'new'.\n");
    continue;
}

/* check if assigning for implementation */
/* if so, store assign-to ids in MR and */
/* authorize for update of affected CIs */
if ((fldidint == 16)
    && (strcmp(newval, "g") == 0))
{
    idend = 0;
    while (idend == 0)
    {
        fprintf(stdout, "Enter id of person to whom MR is");
        fprintf(stdout, " assigned for implementation,\n");
        fprintf(stdout, "          or\n");
        fprintf(stdout, "          carriage return if no more ids");
        fprintf(stdout, " to be entered.\n");
        fgets(loginid, MAXINOPT, stdin);
        if ((cptr = index(loginid, '\n')) != 0)
            *cptr = NULL;
        if (strlen(loginid) == 0)
            idend = 1;
        else
        {
            /* store assign-to id in MR */
            idstored = 0;
            for (j = 1; j < 10; j++)
            {

```

```

    charj[0] = j + '0';
    charj[1] = '\0';
    strcpy(asgnto, "asgnto");
    strcat(asgnto, charj);
    strcat(asgnto, "\0");
    mrrecptr = mrrec;
    for (i = 0; i < NUMKW; i++)
    {
        if (strcmp(mrrecptr -> keyword, asgnto) == 0)
        {
            if (strcmp(mrrecptr -> value, "\0") == 0)
            {
                strncpy(mrrecptr -> value, loginid, MAXVAL);
                idstored = 1;
            }
            break;
        }
        mrrecptr++;
    }
    if (idstored == 1)
        break;
}
if (idstored == 0)
{
    fprintf(stdout, "Cannot store more assign-to");
    fprintf(stdout, " ids in MR.\n");
    idend = 1;
}
}
}

/* store all assign-to ids in table */
for (j = 1; j < 10; j++)
{
    charj[0] = j + '0';
    charj[1] = '\0';
    strcpy(asgnto, "asgnto");
    strcat(asgnto, charj);
    strcat(asgnto, "\0");
    mrrecptr = mrrec;
    for (i = 0; i < NUMKW; i++)
    {
        if (strcmp(mrrecptr -> keyword, asgnto) == 0)
        {
            strcpy(asgntoid[j-1], mrrecptr -> value);
        }
        mrrecptr++;
    }
}

/* get ids of CIs affected by MR and */

```



```

/* authorize CIs for update by assign-to ids */
idend = 0;
while (idend == 0)
{
    fprintf(stdout, "Enter name of CI affected by");
    fprintf(stdout, " implementation of this MR,\n");
    fprintf(stdout, "          or\n");
    fprintf(stdout, "          carriage return if no more");
    fprintf(stdout, " CIs to be entered.\n");
    fgets(ciid, MAXINOPT, stdin);
    if ((cptr = index(ciid, '\n')) != 0)
        *cptr = NULL;
    if (strlen(ciid) == 0)
        idend = 1;
    else
    {
        idstored = 0;
        for (j = 1; j < 10; j++)
        {
            charj[0] = j + '0';
            charj[1] = '\0';
            strcpy(ci, "ci");
            strcat(ci, charj);
            strcat(ci, "\0");
            mrrecptr = mrrec;
            for (i = 0; i < NUMKW; i++)
            {
                if (strcmp(mrrecptr -> keyword, ci) == 0)
                {
                    if (strcmp(mrrecptr -> value, "\0") == 0)
                    {
                        strncpy(mrrecptr -> value, ciid, MAXVAL);
                        idstored = 1;
                    }

                    /* authorize CI for update */
                    for (k = 0; k < 9; k++)
                    {
                        if (strcmp(asgntoid[k], "\0") != 0)
                        {
                            if (authupdt(asgntoid[k], ciid, "a") == -1)
                            {
                                fprintf(stdout, "Authorization failed");
                                fprintf(stdout, " for assign-to");
                                fprintf(stdout, " %s\n", asgntoid[k]);
                                fprintf(stdout, "Please notify");
                                fprintf(stdout, " MR System");
                                fprintf(stdout, " Administrator.\n");
                            }
                        }
                        else
                        {
                            fprintf(stdout, "Authorization completed");
                        }
                    }
                }
            }
        }
    }
}

```

```

        fprintf(stdout, " for assign-to");
        fprintf(stdout, " %s\n", asgntoid[k]);
    }
}
}
break;
}
mrrecptr++;
}
if (idstored == 1)
    break;
}
}
if (idstored == 0)
{
    fprintf(stdout, "Cannot store more CI ids in MR.\n");
    idend = 1;
}
}
}

/* check if changing status to completed or closed */
/* if so, remove CI update authorization */
if ((fldidint == 16) &&
    ((strcmp(newval, "m") == 0) || (strcmp(newval, "c") == 0)))
{
    /* changing status to completed or closed */

    /* store assign-to ids in table */
    for (j = 1; j < 10; j++)
    {
        charj[0] = j + '0';
        charj[1] = '\0';
        strcpy(asgnto, "asgnto");
        strcat(asgnto, charj);
        strcat(asgnto, "\0");
        mrrecptr = mrrec;
        for (i = 0; i < NUMKW; i++)
        {
            if (strcmp(mrrecptr -> keyword, asgnto) == 0)
            {
                strcpy(asgntoid[j-1], mrrecptr -> value);
                break;
            }
            mrrecptr++;
        }
    }

    /* determine what affected CIs are specified in MR */

```

```

/* and remove authorization for assign-to ids      */
for (j = 1; j < 10; j++)
{
    charj[0] = j + '0';
    charj[1] = '\0';
    strcpy(ci, "ci");
    strcat(ci, charj);
    strcat(ci, "\0");
    mrrecptr = mrrec;
    for (i = 0; i < NUMKW; i++)
    {
        if (strcmp(mrrecptr -> keyword, ci) == 0)
        {
            if (strcmp(mrrecptr -> value, "\0") != 0)
            {
                strcpy(ciid, mrrecptr -> value);
                for (k = 0; k < 9; k++)
                {
                    if (strcmp(asgntoid[k], "\0") != 0)
                    {
                        if (authupdt(asgntoid[k], ciid, "e") == -1)
                        {
                            fprintf(stdout, "De-authorization failed");
                            fprintf(stdout, " for assign-to");
                            fprintf(stdout, " %s", asgntoid[k]);
                            fprintf(stdout, " for CI named");
                            fprintf(stdout, " %s\n", ciid);
                            fprintf(stdout, "Please notify MR System");
                            fprintf(stdout, " Administrator.\n");
                        }
                        else
                        {
                            fprintf(stdout, "De-authorization");
                            fprintf(stdout, " completed");
                            fprintf(stdout, " for assign-to");
                            fprintf(stdout, " %s", asgntoid[k]);
                            fprintf(stdout, " for CI named");
                            fprintf(stdout, " %s\n", ciid);
                        }
                    }
                }
            }
            break;
        }
        mrrecptr++;
    }
}

/* make change to field */
mrrecptr = mrrec;

```

```

for (i = 0; i < NUMKW; i++)
{
    if (strcmp(mrrecptr -> keyword, keywdin[keywdid]) == 0)
    {
        if (strcmp(newval, "DELETE") == 0)
        {
            /* make field null */
            strncpy(mrrecptr -> value, "\0", MAXVAL);
            break;
        }
        else
        {
            /* move in new value */
            strncpy(mrrecptr -> value, newval, MAXVAL);
            break;
        }
    }
    mrrecptr++;
}

/* if changing status, update status date */

if ((fldidint == 16) && (strcmp(newval, "i") == 0))
/* assigned for investigation */
{
    mrrecptr = mrrec;
    for (i = 0; i < NUMKW; i++)
    {
        if (strcmp(mrrecptr -> keyword, "invdt") == 0)
        {
            strncpy(mrrecptr -> value, currdate, MAXVAL);
            break;
        }
        mrrecptr++;
    }
}

if ((fldidint == 16) && (strcmp(newval, "a") == 0))
/* approved for implementation */
{
    mrrecptr = mrrec;
    for (i = 0; i < NUMKW; i++)
    {
        if (strcmp(mrrecptr -> keyword, "appdt") == 0)
        {
            strncpy(mrrecptr -> value, currdate, MAXVAL);
            break;
        }
        mrrecptr++;
    }
}

```

```

if ((fldidint == 16) && (strcmp(newval, "r") == 0))
/* rejected */
{
    mrrecptr = mrrec;
    for (i = 0; i < NUMKW; i++)
    {
        if (strcmp(mrrecptr->keyword, "rejdt") == 0)
        {
            strncpy(mrrecptr->value, currddate, MAXVAL);
            break;
        }
        mrrecptr++;
    }
}

if ((fldidint == 16) && (strcmp(newval, "d") == 0))
/* deferred */
{
    mrrecptr = mrrec;
    for (i = 0; i < NUMKW; i++)
    {
        if (strcmp(mrrecptr->keyword, "defdt") == 0)
        {
            strncpy(mrrecptr->value, currddate, MAXVAL);
            break;
        }
        mrrecptr++;
    }
}

if ((fldidint == 16) && (strcmp(newval, "g") == 0))
/* assigned for implementation */
{
    mrrecptr = mrrec;
    for (i = 0; i < NUMKW; i++)
    {
        if (strcmp(mrrecptr->keyword, "asgndt") == 0)
        {
            strncpy(mrrecptr->value, currddate, MAXVAL);
            break;
        }
        mrrecptr++;
    }
}

if ((fldidint == 16) && (strcmp(newval, "m") == 0))
/* implementation complete */
{
    mrrecptr = mrrec;
    for (i = 0; i < NUMKW; i++)

```

```

        {
            if (strcmp(mrrecptr -> keyword, "compdt") == 0)
            {
                strncpy(mrrecptr -> value, currddate, MAXVAL);
                break;
            }
            mrrecptr++;
        }
    }

    if ((fldidint == 16) && (strcmp(newval, "c") == 0))
    /* closed */
    {
        mrrecptr = mrrec;
        for (i = 0; i < NUMKW; i++)
        {
            if (strcmp(mrrecptr -> keyword, "closdt") == 0)
            {
                strncpy(mrrecptr -> value, currddate, MAXVAL);
                break;
            }
            mrrecptr++;
        }
        fprintf(stdout, "Change accepted.\n");
    }
    else
    {
        fprintf(stdout, "\n");
        fprintf(stdout, "MR does not exist.\n");
        fprintf(stdout, "\n");
    }
}

}

/*****
/*  end updatemr.c
*****/

```

```

/*****
/*  valdatmr.c — validate Modification Requests */
*****/

#include <stdio.h>
#include <errno.h>
#include <strings.h>
#include "mr.global.defs"
#include "openmr"
#include "closemr"
#include "callsccs"
#include "checkmr"
#include "getmr"
#include "pathmr"

main(argc,argv)

int argc;
char *argv[];

{
    FILE *valnumfp;

    char *cptr;

    char mrnum[MAXVAL];
    char mrnumlist[MAXVAL];
    char valnum_file[50];
    char fileflag[MAXINOPT];

    int eofflag;

    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "\n");
    fprintf(stdout, "*****\n");
    fprintf(stdout, "This routine validates Modification Request.\n");
    fprintf(stdout, "\n");
    for (;;) /* this causes an infinite loop */
    {
        strncpy(mrnumlist, "\0", MAXVAL);
        fprintf(stdout, "Do you want to read MR numbers from a file?\n");
        fprintf(stdout, "Enter y or yes to read numbers from a file,\n");
        fprintf(stdout, "          or\n");
        fprintf(stdout, "          n or no to enter numbers from the");
    }
}

```

```

fprintf(stdout, " terminal,\n");
fprintf(stdout, "      or\n");
fprintf(stdout, "      carriage return to exit");
fprintf(stdout, " from validate function.\n");
fprintf(stdout, "\n");
fgets(fileflag, MAXINOPT, stdin);
if ((cptr = index(fileflag, '\n')) != 0)
    *cptr = NULL;
if (strlen(fileflag) == 0)
{
    fprintf(stdout, "Validate function terminated by user.\n");
    exit(0);
}
if ((strcmp(fileflag, "y") == 0) ||
    (strcmp(fileflag, "yes") == 0))
{
    fprintf(stdout, "Enter filename of file containing");
    fprintf(stdout, " MR numbers,\n");
    fprintf(stdout, "      or\n");
    fprintf(stdout, "      carriage return to restart validate");
    fprintf(stdout, " function.\n");
    fgets(valnum_file, MAXINOPT, stdin);
    if ((cptr = index(valnum_file, '\n')) != 0)
        *cptr = NULL;

    /* check if valnum file was entered */
    if (strlen(valnum_file) != 0)
    {

        /* open valnum file */
        if (openmr(valnum_file, "r") == -1)
        {
            fprintf(stdout, "Can't open %s file.\n", valnum_file);
            fprintf(stdout, "Check to be sure it exists.\n");
            continue;
        }
        valnumfp = mrfp;

        /* read numbers from file */
        eofflag = 0;
        while (eofflag == 0)
        {
            if (fgets(mrnum, MAXFIELD, valnumfp) != NULL)
            {
                if ((cptr = index(mrnum, '\n')) != 0)
                    *cptr = NULL;
                fprintf(stdout, "File contains mrnum  %s\n", mrnum);
                strcat(mrnumlist, mrnum);
                strcat(mrnumlist, " ");
            }
            else

```



```

        eofflag = 1;
    }
}
else
    continue;
}
else
{
    /* read numbers from terminal */
    fprintf(stdout, "Enter mrnum or carriage return.\n");
    fgets(mrnum, MAXFIELD, stdin);
    if ((cptr = index(mrnum, '\n')) != 0)
        *cptr = NULL;
    while (strlen(mrnum) != 0)
    {
        strcat(mrnumlist, mrnum);
        strcat(mrnumlist, " ");
        fprintf(stdout, "Enter another mrnum or carriage return.\n");
        fgets(mrnum, MAXFIELD, stdin);
        if ((cptr = index(mrnum, '\n')) != 0)
            *cptr = NULL;
    }
}
strcat(mrnumlist, "\n");

/* call checkmr to check if numbers exist */
if (checkmr(mrnumlist) == -1)
    fprintf(stdout, "Modification Request(s) not found.\n");
else
    fprintf(stdout, "All Modification Requests exist.\n");
}
}

/*****
/* end valdatmr.c */
*****/

```

A SYSTEM OF AUTOMATED TOOLS
TO SUPPORT CONTROL OF SOFTWARE DEVELOPMENT
THROUGH SOFTWARE CONFIGURATION MANAGEMENT

by

MARTHA GEIGER WALSH

B. S., Muhlenberg College, 1969

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985

For most software development efforts to be considered successful, certain requirements must be met. These product and project criteria include user satisfaction with product integrity and management objectives to produce the product on time and within budget. In order to meet these requirements, software must be developed in a controlled environment. The discipline called Software Configuration Management can be applied to support controlled development of software.

This report documents a project whose objective was to define, design, implement and document a system of automated tools that would support the control of software development through the application of certain principles of Software Configuration Management, specifically Configuration Control and Configuration Status Accounting. The report includes a review of current literature on Software Configuration Management, specifically relating to control of software development, and a discussion of the requirements, design and implementation of the system of automated tools developed in the project.

The documented automated tools together form a Modification Request System. The functions available include creation of Modification Requests (MRs), update of MRs, selection of certain MRs based on user supplied criteria, listing of MRs, checking for valid MR numbers and authorization to update specific configuration items. Functions are executed by selection from a menu displayed by a supervisor program, which presents a simple interface for the user.