Copyright © 1983 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

Software Development System

By B. R. ROWLAND and R. J. WELSCH

(Manuscript received March 22, 1982)

The 3B20D Processor software development system is an integrated collection of tools and procedures that is used in the development and administration of all 3B20D Processor software. This article describes the tools and procedures and their use in developing the 3B20D Processor software. These tools and procedures include compilers, assemblers, and loaders, as well as change-administration and load-building procedures. The most important characteristic of the development system is the balance between the enforcement of the project standards and the flexibility offered to developers.

I. INTRODUCTION

The software that comprises the operating system, diagnostics and fault recovery, configuration data base, field utilities, and craft interface for the 3B20D Processor has been undergoing development and change over several hundred developer years. Without a strict change-administration strategy and a productive development environment, the tight schedules and reliable deliveries characterizing this system would not have been possible.

This paper presents a description of the software-development environment used by the project's programmers and those administrating 3B20D Processor subsystems. Without the administrative control, these developers might otherwise be limited to simply compiling or assembling programs and attempting to test software with an ad hoc version of the total system software. The emphasis presented here is that of the roles of those involved in the software change process, the tools they each use, the flexibility offered by the tools and administration structure, and the standards and strategies enforced in the development environment.

II. THE ADMINISTRATION OF SOFTWARE DEVELOPMENT

The 3B20D Processor project started with relatively small teams of engineers, compilation tools, and a primitive testing environment. The project has grown to a size that now requires considerable machine support, sophisticated tools, and administrative control. Over 100 developers are responsible for nearly 8,000 source files resident across eight support computers. Compilation of all of these files into a full load takes over 24 hours of machine time on a commercial 16-bit minicomputer. This large amount of software has been partitioned into 23 logically manageable subsystems, the basis for administration and testing.

The 3B20D Processor is the target for output from a software development and administrative environment that resides on the host support computers. All object code for the target is archived, edited, and cross-compiled on the host and then transported by tape or data link for testing on target 3B20D Processors. This separation of host and target ensures a stable development environment as the target machine evolves with new features and performance improvements that may consist of changes in hardware, software, or firmware.

The software administration and development strategy can best be described as one of well-defined and closely tracked data movement between nodes on development host machines and it requires a specific scenario dictating when the data movement takes place. The data are requests for software changes and program files, and the nodes are instances of the software structure reflecting some portion of the total software in one of several development states. Development activity is spread across the host machines by partitioning the software into nearly independent subsystems.

In the development environment, a file system structure containing copies of controlled software source and generated object is referred to as a *node*. When a node is populated with source, object, and target products—all in a like state of development or approval—the node is called a *view* of the software. For each major release, the development system supports three views of the software.

(*i*) The *official* view, or node, contains all of the official source, object, and target products released to customers.

(*ii*) The *approved* view contains those source, object, and target products that have been released as emergency corrections to the last major release. The approved view is an incremental addition to the official view.

(*iii*) The *under-test* view contains those source, object, and target products that will be sent out with the next major release. The under-test view contains everything from the approved view and is otherwise an increment from the combination of the official and approved views.

A final view, that of the developer, may include a mixture of the above views in combination with privately modified versions of various files undergoing a development change.

The approval of a software change submitted by a developer involves the successful integration of that change into a known stable software base. The data movement and approval strategies have a common thread: the incremental modification of a known good software base to produce the next iteration of that base.

An important decision made in the design of the 3B20D Processor development environment was the centralization of the activities used to construct the collection of libraries and objects for installation (i.e., the *load*) on a machine isolated from development activity. This allowed efficient load-building techniques to be developed that do not interfere with developer activity. The implication of centralized load building is that developer changes must be collected and moved to a common point where official load building takes place. In fact, it is only source changes that are moved for load-building purposes. All changes to products are then reconstructed from changed source. Results are then redistributed back to the appropriate development machines.

III. MAJOR SOFTWARE DEVELOPMENT SYSTEM COMPONENTS

It is the interaction of four distinct software administration and generation systems that creates the 3B20D Processor Software Development System (SDS). This section presents an overview of these systems; Section IV describes their use; and Section V describes the major standards that contribute to the environment and are enforced by the project's tools.

3.1 The Modification Request System

The Modification Request (MR) data base system is a general-purpose hierarchical system tailored to 3B20D Processor change-tracking requirements. The MR is the entity that identifies all requests for 3B20D Processor software changes and tags all source changes made by developers and administrators to official 3B20D Processor software. The MR system serves as a central master data base from which administrators control and track all software changes and generate appropriate tracking and status reports.

As implemented for this project, the MR data base is actually a three-level data base comprised of an MR level, one or more release levels per MR, and one or more subsystem levels per release. The higher the level (the MR level is highest), the more global the scope of the information maintained.

The MR level information includes an identification number, originator information, problem description, priority, severity, and due date. At the release level, the data base includes due date, feature engineer and developer, and priority. Subsystem level data includes responsible developer, modified source files, status, solution description, and load-building instructions. The majority of the information is automatically generated and stored in response to developer and administrator actions. For the most part, direct data entry is not necessary.

The MR system includes a high-level query language and report-formatting facilities for producing reports from the data base. The query language is very powerful in that it allows for selection specification and sorting levels. The reporting facilities process the output of the query language. Additional capabilities exist that cause brief reports, called synopsis reports, to be generated automatically and sent to appropriate administrators, supervisors, and/or developers in response to particular status changes. For example, when an MR requires a change in a particular subsystem, the responsible developer receives a synopsis report. When the responsible developer submits changes in response to an MR for system test, appropriate administrative personnel receive synopsis reports.

3.2 The Change Management System

The Change Management System (CMS) is a facility compatible with the *UNIX** operating system aimed at controlling the activities of both developers and administrators related to software change and change approval. The CMS uses the *UNIX* operating system Source Code Control System (SCCS) and its own relational data base to track and relate each change made in a source file to an MR.[1]

All official 3B20D Processor source files are maintained via CMS on a per-subsystem basis. A CMS instance is defined to be a set of SCCS-controlled source files and the relational data base tracking the MR-based changes. Each subsystem is a unique instance of CMS. There are, therefore, one or more CMS instances on each development machine (see Fig. 1). This strategy was chosen over a strategy having only one CMS instance per machine for the following reasons:

(*i*) Corruption of a CMS data base would only affect one subsystem.

(*ii*) The majority of the developers need access to the official source files for only one subsystem.

(*iii*) Separate CMS instances per subsystem allow load balancing on the development machines to be more easily implemented. Instead of having to extract a portion of a large data base and move it with developers to another machine, an entire CMS data base can be moved.

* Trademark of Bell Laboratories.

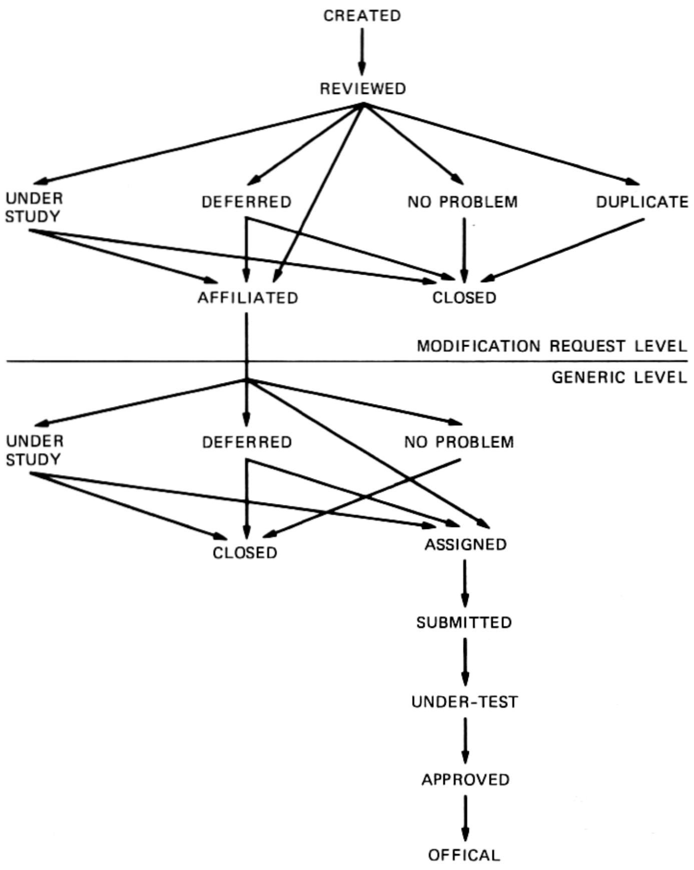

Fig. 1—Diagram of the state changes in the Modification Request System.

3.3 The Software Generation System

The software development system is geared to construct executable software for the following three target environments:

(*i*) A 3B20D Processor running the DMERT operating system.[2] This is the primary target for official system software construction.

(*ii*) A PDP 11/70 running a real-time version of the *UNIX* operating system. This is the recognized 3B20D test laboratory support processor.

(*iii*) A PDP 11/70 running the *UNIX* operating system. This is the primary machine and operating system for development and official software construction.

The primary target for which the SDS generates code is the 3B20D Processor. A Software Generation System (SGS) containing a compiler, assembler, and link editor is provided for generating and supporting 3B20D Processor binary executable object modules on the development and load-building machines.

3.4 Building software views

The major tool used by both developers and administrators in conjunction with the SGS for generating 3B20D Processor loads is the build tool. This tool invokes the minimum number of commands necessary to create a requested object incorporating selectable states of related software from its components. It determines, according to a specified set of dependency information, exactly which objects must be rebuilt (because they may be out of date) and which objects are current with respect to the changes desired in the requested object to avoid unnecessary compilation steps. This dependency information is automatically generated by a utility from the software components. This ability to create an object with particular versions of software that reflect its state or submitted MR software changes is known as *building a software view.*

IV. THE SOFTWARE DEVELOPMENT PROCESS

All software in the DMERT environment is created, corrected, or enhanced in conjunction with an MR. With this association of an MR to every software introduction or change, all development on the project can be adequately monitored. The assignment of an MR allows a developer to make modifications to official project source and to ultimately submit the changes for approval by project administrators. The development activities that occur and the tools used in the process between the assignment of an MR and the submission of changes are described in this section.

4.1 MR handling

The MR system maintains modification requests in a variety of states (see Fig. 1) that reflect the activity being taken on the request. An MR is *created* by other developers or applications and then initially *reviewed* by administrators. It may be found that an MR is a *duplicate* of a previously resolved MR; it may be required to be placed *under study*; it can be *deferred* for later consideration; or it may be found to be *no problem* in which case the MR is ultimately *closed.* The under study or deferred MRs eventually will be either *closed* or *affiliated* with generics for action. Once the MR level state is affiliated, then generic level action is allowed.

MR assignment is made to one or more subsystems, as many as are required to implement a solution. When an MR is assigned, a developer then creates a development node in which to maintain copies of source to be created or modified in response to the modification request and to build a view of the subsystem affected by the MR. This private development node gives the developer the freedom to make any changes with any set of tools on local copies of source files. The developer is now free to experiment with solutions to satisfy the MR, while official source remains protected by CMS. Source files intended to be modified by the developer for the MR are requested via CMS commands that associate the changes with the MR and the release to which the MR is being applied. The association of MR to source files changed is kept in the CMS data base to allow administrators or other developers to request versions of software containing solutions to particular problems. While a developer has copies of official source to be worked on in a private node, all other developers are prevented by CMS from obtaining copies of the same source to avoid creating unsynchronized changes. Using private source copies, the developer is ready to make changes and create new subsystem objects for testing.

4.2 Languages and tools

Software for the 3B20D Processor target is written primarily in the high-level programming language C with occasional use of the 3B20D Processor assembly language. The C language contains many modern control and data structures found in languages such as PASCAL. It is characterized by its brevity of expression, direct access to data type representation, and operations and declarations available to the programmer to enhance the generation of efficient assembly level code. The 3B20D Processor assembly language is a member of IS25, a 3B20D family standard specifying:

(*i*) Activation stack format

(*ii*) Data type representations

(*iii*) Registers

(*iv*) Operations and addressing modes for accessing and manipulating data objects ranging in size from a single bit to 32-bit words.

The tools used to compile C programs and assemble assembly code into user level objects are modeled after the tools used with the *UNIX* operating system and for the development of the *BELLMAC**-8 microprocessor.[3] An additional complement of tools is used to create and modify special process files and prepare the developer with information that will be very valuable in the testing environment.

A single command can be used to control the C program compilation process. This command invokes a source code preprocessor, the C compiler and optimizer, assembler, and link editor. These four tools in turn convert a collection of C programs into a single object with addresses that are either relocatable or absolute (Fig. 2).

* Trademark of Western Electric.

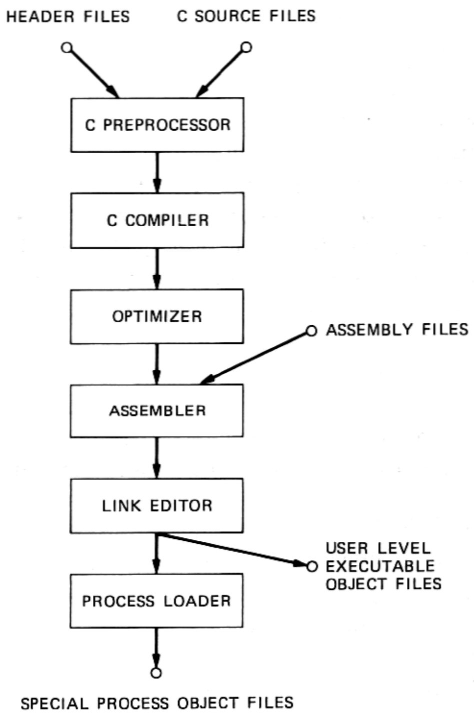

Fig. 2—Compilation tools.

The preprocessor provides a macro expansion facility and directives for sharing common source (or *header*) files. Header files typically contain data and macro declarations shared by many programs. This mechanism ensures that common definitions are consistent across all programs using them because they are accessed from the same source.

The C compiler and optimizer are based on the portable C compiler[4] and a similarly structured portable code improver. The compiler generates assembly code from preprocessed C source using syntax-directed parsing (from *YACC*[5]) and modified Sethi-Ullman[6] and Aho-Johnson[7] tree-matching and expression-optimization techniques. The generated assembly code can then optionally be passed through an optimizer (or, more correctly, code improver) that eliminates unnecessary branching, removes redundant register loads, and converts certain instructions or sequences of instructions into simpler or more efficient, yet semantically equivalent ones. Both the compiler and optimizer leave their result in the form of an assembly language file.

From assembly language source, the assembler creates an object file containing object code text and data (with optional relocation information), symbol and source line number tables to communicate with testing tools, and a section layout dictionary that provides information on the structure of the file.

The link editor is capable of combining a collection of object files into a single object by resolving interprogram symbol references and binding symbols to virtual or absolute addresses as specified. The object file for the 3B20D Processor may contain multiple text and data sections. This facility is used in the creation of special process files and libraries.

Process files for the DMERT operating system are generated using a special tool that uses the link editor to create special data sections used in process communication and entry. All kernel, supervisor, and special DMERT processes require use of this tool as a final construction step. User level processes require only the normal link edit step for completion.

4.3 Building a view of a subsystem

The object construction tool, build, (based on the *UNIX* operating system make command), exists to create objects according to a previously specified set of object construction commands and a list of dependencies. This specification is used to identify which software components are needed to construct others.

Using modification time-stamps provided by the file system, build determines which objects are current and which need to be rebuilt. When an object needs to be rebuilt, its specification must be examined, and so on down the line until all necessary objects are rebuilt or current. These object construction instructions constitute a makefile for the particular DMERT subsystem.

The advantage that build offers over make is that the developer in a private development node need only have copies of the source being changed and can access the remainder of the objects, header files, or C or assembler source files from other nodes as specified in the developer's *viewpath.* A viewpath is an ordered set of directory names indicating the nodes to be searched for missing software components. The viewpath is an extension of the *UNIX* operating system *searchpath* variable that contains a list of directories containing commands. By specifying a viewpath, a developer can create a desired view of a subsystem product to be tested. The view may be, for example, that of the developer's changes integrated with only officially approved software or with any other software currently submitted to the test team.

4.4 Preparation for testing

Once a subsystem has been changed in response to an MR within a developer's node and is ready to be taken into the laboratory for testing, there are listings that can be prepared to aid in the software testing process. A set of SGS utilities exist to generate these listings. A C program breakpoint source listing contains a source listing of all the functions in a file augmented by line numbers indicating C source lines at which a breakpoint can be set during testing. This listing is important due to the fact that object code may have been significantly rearranged during the optimization step of the compiler rendering certain C source lines shuffled in a semantically equivalent, but non-obvious, fashion. The listing helps a developer in a test laboratory who needs to correlate high-level C source to 3B20D Processor assembly code.

A namelist utility creates a listing of all the C language symbols residing in the object file's symbol table with their addresses and types. Not all symbols associated with an object need reside in the symbol table since they can be removed during object creation by another SGS utility.

A developer can obtain assembly source listings associated with an object file in either of two ways. The listing can be generated by the compiler during compilation or it can be created by a *disassembler* from the object file. This latter listing will contain C program source information in terms of symbols, labels, and line numbers if they have not been stripped from the object. This listing can be of particular interest after software has executed on the machine to determine where text and data may have been accidently altered by a process out of control in a development laboratory.

With these listings augmenting the original source files, the developer is prepared to enter the testing environment.

4.5 Submitting changed files for approval

When all desired changes have been made and unit or subsystem testing has convinced the developer that the modification request has been satisfied, the files extracted from the official CMS data base (and any files newly created as a result of the MR) are submitted to the test team through the use of CMS commands (refer again to Fig. 1). The activity of the developer on the files in question can be temporarily or permanently suspended by other CMS commands that place the changes made into the CMS data base and allow other developers access to the same files for editing if necessary.

The software approval process in the SDS is initiated when the developer submits an MR-related software change for inclusion in the next release. The developer submittal begins a three-step approval process:

(*i*) Independent Certification—The source change is used by administrators to reconstruct the changed products. This is done in such a way as to not affect anything currently approved or under test. The changed products are then independently tested (certified) by a member of the project's system test group. Certification failure implies rejection of the MR and further work for the developer.

(*ii*) Integration—Those changes passing certification are incorporated into the under-test view where they are integrated with the rest of the under-test changes. This integration is performed by members of the project's integration group. Failure during integration also implies MR rejection, as well as recertification following additional development.

(*iii*) Approval—When all certified changes have been integrated and soak tested, the under-test view will be approved. This implies releasing a software update to 3B20D customers, updating all the nodes, and approving MRs.

V. DEVELOPMENT ENVIRONMENT AND STANDARDS

The software development system must establish a balance between the flexibility given to developers and administrators to enhance their productivity and restrictive standards so that administration is manageable and effective. Considerable development flexibilities already have been identified. Among these are:

(*i*) The capability to select a particular version of a source file by specifying one or more MR numbers whose related source changes are applied to the last released version of the source file.

(*ii*) The use of any editing facilities once a source file has been obtained from CMS.

(*iii*) The C language, which encourages programmer optimization and functional modularization.

(*iv*) The protection from concurrent source changes by independent developers ensured by CMS.

(*v*) The independent, private development node allowing experimentation that does not interfere with other development activity.

(*vi*) The flexible capabilities for view construction provided by build.

(*vii*) The ability administrators have through CMS to back out changes associated with particular MRs and return to previous software states.

The software environment and its SDS tools also impose an important set of standards on development activities that guarantee safe, productive, and orderly administration of all change activity.

5.1 Host machine configuration

The host machines used in software development and administration are each standardized with respect to the activities that occur on the machines. This standard ensures that all development activity can be monitored adequately and that the proper tools can be made available to those needing them. With a fixed set of machines, updates to the software tools can be synchronized making sure that all development is compatible.

The 3B20D Processor software development system resides on a network of eight general-purpose, 16-bit minicomputers, all of which run the *UNIX* operating system. The functional configuration of these eight machines is shown in Fig. 3.

(*i*) Developer activity is confined to the six machines labeled SD (software development). Here developers utilize CMS, the SGS, and the build facility, as well as the standard tools available with the *UNIX* operating system.

(*ii*) The machine labeled SC (software control) is for administrative use only. This machine contains the modification request data base system from which all change activity is controlled.

(*iii*) The SP (software production) machine is an administrative machine. All official product construction takes place on this machine.

The test environment combines a 3B20D Processor and a support machine running a real-time version of the *UNIX* operating system as its support processor.

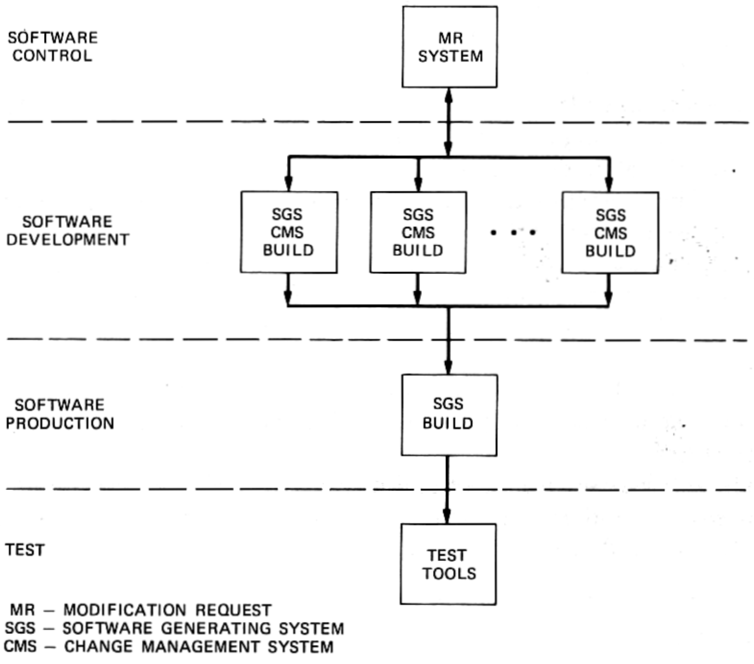

Fig. 3—Host machine configuration.

5.2 Software configuration

A standard *UNIX* operating system directory structure configuration has been established for the 3B20D Processor software subsystems. This configuration is repeated in each CMS node so that file access and object construction techniques are fully repeatable no matter where the view building activity takes place.

The 3B20D software is logically partitioned into distinct subsystems. Typical subsystems include the operating system, diagnostics, and peripheral unit drivers. Software development for the subsystems is evenly spread over the six SD machines described in the previous section. A single subsystem is fully contained on one of the SD machines. Each SD contains the software development activities for one or more subsystems.

The only machine on which the full directory structure combining all of the subsystems exists is the SP machine where official load construction takes place. Subsystem development can take place on the individual SD machines, independent of other subsystems, because the following software construction standard is enforced: the only source files shared between subsystems during load construction are global header files (common definitions required by more more than one subsystem) or global libraries (collections of common object modules).

Global header files and global libraries have fixed, standard positions in the SD machines' directory structure. These shared resources are distributed from the SP machine to the various SD machines after validation by administrators. In this way, all SD machines are kept synchronized with respect to this critical data.

Other top level directories or directory structures are used during the load-construction process as installation points for revised tools, tool usage descriptions in manual page format, and 3B20D Processor core software components.

5.3 Interactions between CMS and the MR system

The major strength of CMS and the MR system lies in the standardization of source-change administration. The following major standards are enforced with these tools:

(*i*) Every incremental change is tagged with an MR number.

(*ii*) The developer cannot modify official source until an MR has been assigned to him or her by an administrator. This assigning capability can be used by project administrators to funnel developer activity.

(*iii*) Once a developer is satisfied with a change, it is submitted against the corresponding MR for independent verification. Once submitted, no further source changes for the MR are allowed.

The MR data base system on the SC machine and the various CMS instances, each identified via a subsystem name, on the six SD machines are connected in the SDS via a high-speed network and a remote job-execution facility. Specific actions on the SC machine cause remote jobs to be sent to one of the SD machines and executed for a particular CMS instance. The reverse is also true. The major triggers interconnecting the MR system and the CMS instances are:

(*i*) When an MR is assigned to a particular subsystem for a particular generic, a remote job is sent to the SD machine on which that subsystem's development is based. When the job is executed on the SD machine, a sequence of CMS commands is executed that cause the MR to be assigned to the particular generic in the correct CMS instance. The developer can then officially edit the source files to effect the change.

(*ii*) When the developer is satisfied with changes, they are submitted for verification. The CMS status is set to *submitted* to prohibit any further editing for this MR, and a remote job is sent to the MR system on the SC machine. This job will cause the MR status for the particular generic and subsystem to change to *submitted.*

(*iii*) A remote job to reject an MR can be sent to CMS on an MR status change of under-test to assigned or submitted to assigned. In CMS, the MR would again be available for editing with a subsystem status of being-fixed.

These are the only MR/CMS interconnections necessary. All other control functions require only MR system actions.

5.4 Product construction standardization

A major ingredient of the 3B20D Processor SDS is the standardization of the format and content of the makefiles used by build. The standards are enforced by allowing developers to create only a skeleton makefile and providing a makefile generator to produce the full makefile. The advantages accrued by standardized makefiles are the guaranteed correctness and completeness of object dependency lists and the ability to create cross-reference listings of dependencies to determine changed file impacts.

VI. SUMMARY

Software development for the 3B20D Processor is administered with an important set of standards and a powerful set of tools to track development efforts and enforce these standards. The developers have at their disposal a flexible system for the generation of C programs and utilities to adequately prepare for debugging and efficiently construct object programs. The resulting 3B20D Processor software development system strikes a critical balance between offering freedom and flexibility to programmers while managing and monitoring orderly software change procedures.

REFERENCES

1. M. J. Rochkind, "The Source Code Control System," IEEE Trans. Software Eng., *SE-1* (December 1975), pp. 364–370.
2. J. R. Kane, R. E. Anderson, and P. S. McCabe, "3B20D Processor & DMERT Operating System: Overview, Architecture and Performance of DMERT," B.S.T.J., this issue.
3. H. D. Rovegno, "A Support Environment for MAC-8 Systems," B.S.T.J., *57,* No. 6 (July–August 1978), pp. 2251–64.
4. S. C. Johnson, "A Portable Compiler: Theory and Practice," Conf. Rec. Fifth Annual ACM Conference on Principles of Programming Languages, Tucson, Arizona, January 23, 1978, pp. 97–104.
5. S. C. Johnson, and M. E. Lesk, "Language Development Tools," B.S.T.J., *57,* No. 6 (July–August 1978), pp. 2155–77.
6. R. Sethi and J. D. Ullman, "The Generation of Optimal Code for Arithmetic Expressions," J. ACM, *17,* No. 4 (October 1970), pp. 715–28.
7. A. V. Aho and J. C. Johnson, "Optimal Code Generation for Expression Trees" J. ACM, *23,* No. 3 (July 1976), pp. 488–501.