HO

UNOS 1556

Bell Laboratories **Cover Sheet for Technical Memorandum**

The information contained herein is for the use of employees of Bell Laboratories and is not for publication (see GEI 13.9-3)

Title: **Source Control + Tools = Stable Systems**

Date: **May 15, 1980**

TM: **80-3168-7**

Other Keywords: **UNIX**
Software Management
Software Tools

Author(s)	Location	Extension
Eugene Cristofor	HO 1E-301	7891
T. A. Wendt	HO 1G-322A	7568
B. C. Wonsiewicz	HO 1E-325	3272

Charging Case: **49408-121**
Filing Case: **40324-2**

ABSTRACT

This paper describes our experience in administering the software for a large system. This task was accomplished with the help of a Software Manufacturing System, that we designed and implemented.

The manufacturing system is based on strictly controlled and identified source files, whose identification is preserved in compiled and loaded products. The identification can be automatically extracted from the product, and forms a complete specification for product manufacture. A hierarchy of products can be specified in this way so that an entire system can be specified by a single label.

The requirements for the manufacturing system were:

The system must be *reproducible* in any version by a third party.
All changes to the software must occur at the source level.
The building process must be initiated by a *single* command, and use a minimum of machine resources.

The techniques described here are based on simple tools applicable to other development environments.

Pages Text:	1	Other:	11	Total:	12
No. Figures:	5	No. Tables:	0	No. Refs.:	9

E-1932-U (3-76) SEE REVERSE SIDE FOR DISTRIBUTION LIST

BELL TELEPHONE LABORATORIES, INC. TM-80-3168-7

DISTRIBUTION
(REFER GEI 13.9-3)

COMPLETE MEMORANDUM TO

CORRESPONDENCE FILES

OFFICIAL FILE COPY
PLUS ONE COPY FOR
EACH ADDITIONAL FILING
CASE REFERENCED

DATE FILE COPY
(FORM E-1328)

10 REFERENCE COPIES

BAUER,R C
BEAUMONT,LELAND R
>BLOSSER,PATRICK A
•BOEHM,EARL W
>BOURNE,STEPHEN R
BREITHAUPT,A R
CICON,J P
DOMPIERRE,J A
DONOHUE,B P,3RD
DOUGHERTY,H J
•DOWDEN,DOUGLAS C
>FELDMAN,STUART I
FREEMAN,K G
GILLON,ALEX C
>GLASSER,ALAN L
HAISCH,R F,JR
HAYDEN,DONALD F,JR
HEIDER,BRUCE H
HERGENHAN,C B
JAASMA,E G
JONES,E E
KAPLAN,FRANK
>KERNIGHAN,BRIAN W
MACRI,P P
MALIK,JOSEPH M
MARIONE,JOHN P
MAUNSELL,B I
MC DONALD,J Y
MCCULLOCH,JOHN W
MITCHELL,J C
OHOHUNDRO,WAYNE E
PASTERNACK,G
PILLA,M A
REBERRY,D L
•REESE,RANDALL D
>RITCHIE,D M
>ROCHKIND,MARC J
RODRIGUEZ,ERNESTO J
•ROSENTHAL,CHARLES W
SHAER,N R
SNYDER,BERNARD E
STOREY,THOMAS F
STREETER,LYNN A
STUBBLEFIELD,R W
>THOMPSON,K
>UNGAR,DAVID M
WELT,M J
WONSIEWICZ,B C
YORK,B I
ZISLIS,PAUL M

COVER SHEET ONLY TO

50 NAMES

COVER SHEET ONLY TO

CORRESPONDENCE FILES

4 COPIES PLUS ONE
COPY FOR EACH FILING
CASE

AAGESEN,JOHN
ACKERMAN,J T
ACKROFF,JOHN M
AHO,ALFRED V
AHRENS,RAINER B
ALBERALLA,RICHARD J
ALCALAY,D
ANDERSON,KATHRYN J
ANDERSON,MILTON M
ANTOLICK,DAVID R
APPULINGAM,SUBRAMANIAM
ARNDT,DENNIS L
ARNOLD,GEORGE W
ARNOLD,JAMES Q
ARNOLD,PHYLLIS A
ARNOLD,THOMAS F
ASTHANA,ABHAYA
AULL,DENIS W
BABECKI,GLENN R
BABU,RAJESH RATILAL
BALENSON,CHRISTINE M
BALLANCE,ROBERT A
BARBATO,ROBERT R
BARCLAY,DAVID K
BAROFSKY,ALLEN
BARON,ROBERT V
BAUER,BARBARA T
BAUER,HELEN A
BAUMAN,STEVEN M
BAVIER,RICHARD J
BEACHY,MILTON
BEBLO,WILLIAM
BECKER,CURTIS A
BECKER,RICHARD A
BECK,WANDA
BEDNAR,JOSEPH A,JR
BEEKMAN,BERNARD
BENCO,DAVID S
BENISCH,JEAN
BENNETT,RAYMOND W
BENNETT,RICHARD L
BENNETT,WILLIAM C
BERGLAND,G D
BERNHARDT,RICHARD C
BERNSTEIN,L
BERRYMAN,R D
BERZINS,ALEXANDER H
BILOWOS,R M
BIREN,IRMA B
BISHOP,THOMAS P

COVER SHEET ONLY TO

BITTNER,B B
BLAKE,GARY D
BLAZIER,S D
BLECHMAN,RONALD I
BLEIER,JOSEF
BLINN,J C
BLISS,ROBERT H
BLUM,MARION
BODEN,F J
BOGART,THOMAS G
BOIVIE,RICHARD H
BOLSKY,MORRIS I
BOPP,J ROBERT
BORDELON,EUGENE P
BORISON,ELLEN A
BOWYER,L RAY
BOYCE,W M
BOYER,PHYLLIS J
BOYLE,GERALD C
BRADLEY,M HELEN
BRADLEY,R H
BRADY,PAUL T
BRAUN,DAVID A
BRIGGS,GLORIA A
BROAD,MARTHA M
BRONSTEIN,N
BRONZO,JOSEPH A
BROOKS,CATHERINE ANN
BROSS,JEFFREY D
BROWMAN,INNA
BROWN,ELLINGTON L
BROWN,LAURENCE MC FEE
BROWN,W STANLEY
BRUECKNER,DOUGLAS E
BURG,F M
BURKE,THERESA M
BURNS,GARY J
BUROFF,STEVEN J
BUROK,DILINA
BUSCH,KENNETH J
BUTLETT,DARRELL L
BYERLEE,R W
BYORICK,ROBERT S
BYRNE,EDWARD R
CAMPBELL,JERRY H
CANADAY,RUDD H
CANDREA,RONALD D
CAREY,F E,JR
CARTER,DONALD H
CASPERS,BARBARA E
CASTELLANO,MARY ANN
CATO,H E
CAVINESS,JOHN D
CERMAK,I A
CHAI,D T
CHAMBERS,B C
CHAMBERS,J M
CHANEY,F W
CHANG,CHUNG W
CHANG,JO-MEI
CHAPPELL,S G
CHENG,Y

COVER SHEET ONLY TO

CHEN,PING C
CHEN,ROBERT
CHEN,STEPHEN
CHERRY,LORINDA L
CHIEN,FU-LI BETTY
CHILDS,CAROLYN
CHIN,AUGUSTIN Y
CHIN,KATHLEEN E
CHI,M C
CHODROW,M M
CHONG,PHEE
CHRIST,C W,JR
CIEMINSKI,DEBRA F
CLARK,DAVID L
CLAYTON,D P
CLEWELL,JAMES L
CLINE,LAUREL M I
COHEN,HARVEY
COHEN,KAREN L
COHEN,NEIL B
COHEN,RICHARD L
<COLE,LOUIS M
COLLICOTT,R B
COLWELL,ELLEN MARIE
CONKLIN,DANIEL L
CONNERS,RONALD R
COOK,JOEL M
COOK,T J
COOPER,ARTHUR E
COSCHIGANO,J J,JR
COSTON,W P
COTTRELL,JENNIE L
COVINGTON,RALPH L
CRAGUN,JOAN
CRAIG,JOHN R
CRISTOFOR,EUGENE
CRUPI,JOSEPH A
DAVEY,DOUGLAS A
DAVISON,JOSEPH W
DAVIS,R DREW
DAWSON,JOHN E
DAY,F W
DE FAZIO,M J
DE GRAAF,D A
DE TREVILLE,JOHN D
DEAN,JEFFREY S
DENNY,MICHAEL S
DENSMORE,SUSAN
DESMOND,JOHN PATRICK
DEVLIN,SUSAN J
DI PIETRO,R S
DIB,GILBERT
DIMMICK,JAMES O
DINEEN,THOMAS J
DOCK,G A
DOEDLINE,BARBARA ANN
DOLATOWSKI,VIRGINIA M
DOLOTTA,T A
DOMANSKI,BERNARD
DOWDEN,IRIS S
DRAKE,LILLIAN
DREIZLER,H K

COVER SHEET ONLY TO

D'ANDREA,LOUISE A
DUDICK,ANTHONY
DUGGER,DONALD D
DUMAIS,VALERIE
DUNCANSON,ROBERT L
DUNKIN,PATRICIA
DURAND,JANIS C
DWYER,T J
DYER,MARY E
DZIK,STEVEN C
EDMUNDS,T W
EICHORN,KURT H
EISELE,RONALD C
EITELBACH,DAVID L
EKSTROM,SUSAN
ELDRIDGE,JOHN
ELLIS,DAVID J
ELY,T C
EMERSON,DOMINIQUE M
ENGLAR,BRUCE WYATT
EPLEY,ROBERT V
ESCOLAR,CARLOS
ESSERMAN,ALAN R
EVERMAN,THOMAS L,JR
FABISCH,M P
FABRICIUS,WAYNE H
FAIRCHILD,DAVID L
FALLON,J M
FAULKNER,ROGER A
FEAY,MARY R
FEDER,J
FELTON,WILLIAM A
FERRER,NANCY L
FEUSTER,I REED
FISCHER,HERBERT B
FISHMAN,DANIEL H
FLANDRENA,R
FLEISCHER,H I
FLEMING,JAMES R
FORSON,HENRY M
FORTNEY,V J
FOUGHT,B T
FOUNTOUKIDIS,A
FOWLER,BRUCE R
FOWLER,GLENN D
FOX,PHYLLIS A
FOY,J C
FRANKLIN,JAMES W
FRANK,AMALIE J
FRASER,A G
FRASER,D L,JR
FREEMAN,MARTIN
FREMON,R C
FRENCH,A F,JR
FRIED,LAWRENCE K
FROST,H BONNELL
FRUCHTMAN,BARRY
GABBE,JOHN D
GALE,ALAN O
GAMA,JORGE L
GARRISON,GARY A
GARST,BLAINE,JR
...
754 TOTAL

• NAMED BY AUTHOR > CITED AS REFERENCE < REQUESTED BY READER (NAMES WITHOUT PREFIX WERE SELECTED USING THE AUTHOR'S SUBJECT OR ORGANIZATIONAL SPECIFICATION AS GIVEN BELOW)

MERCURY SPECIFICATION..

COMPLETE MEMO TO:
316-SUP

COVER SHEET TO:
316-TA

COPRSM = COMPUTER PROGRAMMING MANAGEMENT

HO CORRESPONDENCE FILES
HO 1A127

TM-80-3168-7
TOTAL PAGES 7

TO GET A COMPLETE COPY:

1. BE SURE YOUR CORRECT ADDRESS IS GIVEN ON THE OTHER SIDE.
2. FOLD THIS SHEET IN HALF WITH THIS SIDE OUT AND STAPLE.
3. CIRCLE THE ADDRESS AT RIGHT. USE NO ENVELOPE.
4. INDICATE WHETHER MICROFICHE OR PAPER IS DESIRED.

PLEASE SEND A COMPLETE

( ) MICROFICHE COPY ( ) PAPER COPY

TO THE ADDRESS SHOWN ON THE OTHER SIDE.

Bell Laboratories

subject: **Source Control + Tools = Stable Systems**
Case: **49408-121**
File: **40324-2**

date: **May 15, 1980**

from: **Eugene Cristofor**
HO 3168
1E-301 x7891

T. A. Wendt *
HO 3167
1G-322A x7568

B. C. Wonsiewicz
HO 3168
1E-325 x3272

TM: **80-3168-7**

MEMORANDUM FOR FILE

The attached paper will be presented at the Fourth International Conference on Computer Software and Applications - COMPSAC '80. The Conference will be held in Chicago, on October 29-31, 1980.

[illegible]
Eugene Cristofor

[illegible]
T. A. Wendt *

[illegible]
B. C. Wonsiewicz

HO-3168-EC/TAW/BCW-nroff

Atts:
"Source Control + Tools = Stable Systems"

* Mr. Wendt was in Dept. 3168 during the time covered in this paper.

Source Control + Tools = Stable Systems

Eugene Cristofor
T. A. Wendt
B. C. Wonsiewicz

Bell Laboratories
Holmdel, New Jersey 07733

ABSTRACT

This paper describes our experience in administering the software for a large system. This task was accomplished with the help of a Software Manufacturing System, that we designed and implemented.

The manufacturing system is based on strictly controlled and identified source files, whose identification is preserved in compiled and loaded products. The identification can be automatically extracted from the product, and forms a complete specification for product manufacture. A hierarchy of products can be specified in this way so that an entire system can be specified by a single label.

The requirements for the manufacturing system were:

The system must be *reproducible* in any version by a third party.
All changes to the software must occur at the source level.
The building process must be initiated by a *single* command, and use a minimum of machine resources.

The techniques described here are based on simple tools applicable to other development environments.

1. INTRODUCTION

This paper discusses software administration for a large software project. By large, we mean hundreds of developers, thousands of source files, and millions of bytes of executable code. The problem addressed is how to efficiently and automatically manage the software, so that any given version of the system can be manufactured and, if necessary, changed.

Control of the system is based on control of the source code. Each source file contains a unique *label*, which is reproduced in objects constructed from the file. A product (object, library, or executable file) contains the *label*s of all the source needed to construct it. Tools have been built to extract the *label*s to form a complete and exact specification for the manufacture of the product. The basic technique supports the manufacture of a hierarchy of products: library, process, subsystem, and system. A single *label* is sufficient to specify any of the above.

The manufacturing system described is based on the use of the UNIX[1] time-sharing system [DMR], and uses the Source Code Control System (SCCS) [MJR] [ALG].

For the sake of simplicity, the case of all developers residing on a single machine is discussed first. The multimachine environment poses special problems that are discussed separately.

2. BASIC ELEMENTS

The *system* to be manufactured consists of several hundred *processes*, or executable entities. The processes may be grouped into *subsystems* characterized by a large number of common interfaces. The processes are built by compiling *source* files into *object* files and then loading them together with *libraries*, which are collections of object files.

The discussion begins with the source file.

1. UNIX is a Trademark of Bell Laboratories.

2.1 File Names

Every source file necessary to manufacture the system is given a unique name. The source files are administered by SCCS which assigns a unique number to each version stored. The name and version number (SCCS id) constitute the *label*, which is represented as an ASCII string. The *label* is all that is needed to retrieve that version of the file from the system archives.

Every file extracted from the archives contains the *label* concatenated with a key string of characters which can be recognized automatically. The *label* is compiled into every object file produced from the source file. Figure 1 depicts the transforms of source files into processes, and the presence of the *label* at every stage.

For example, if a C language [BWK] source file named *copy.c* at version 1.7 were compiled, placed in a library, and subsequently loaded into a process, the source file, the object file, the library, and the process would all contain the *label*, "copy.c 1.7".

Further, if during compilation, the source file *copy.c* included[2] two other files *x.h* and *y.h*, their *labels* would be included in all the previous objects. Since all dependencies are tracked in this way, the typical process will contain hundreds of *labels*.

The SCCS *what* command extracts the *labels* from an object, be it a source file, an object file, or a process. Figure 2 shows how the *what* command works.

For example, the output of the *what* command on the object file *copy.o* will look like this:

```
copy.c  1.7
x.h     2.1
z.h     5.1
```

2.2 SLISTS - Product Table of Contents

The output of *what* can be captured as the full description of the source files required for the process. We call it an *slist*, for SCCS id list. The *slist* is the complete and precise specification of the source files needed to make an object. Since it is a file, it may itself be an SCCS file with its own *label* [ALG1].

2. The C language supports a file inclusion mechanism.

If the *slists* for products *a* and *b* are *a.sl* and *b.sl*, the subsystem containing them could be described by another slist *ab.sl*:

```
a.sl   7.21
b.sl   1.1
```

The subsystem *slist* has its own *label* (name and version), and could be an element in another *slist* in the next level of the hierarchy. Ultimately, the entire system specification can be captured in a single *slist*, referred to as the *master slist*[3]. The specification may be hierarchical, by referencing a hierarchy of *slists*. The hierarchy can be changed by changing the content of the *slists*. The hierarchy of slists for the example given in this Section appears in Figure 3.

It is possible to check the consistency of files referenced in the *slist*. If the same file is referenced with different versions, the *slist* is inconsistent. This might arise if a program were compiled with a new version of a shared data file, but loaded with a library compiled with an older version. The inconsistency may be either malignant or benign, we usually assume the worst.

The systematic description of the files in a system has other benefits. A similar check can be made to the *slist* hierarchy to verify the consistency of the entire system. The impact of a proposed change in a single source file can be estimated by counting the number of references to the respective file in the *slists*. Finally, the objects built from the *slist* specification can be checked by comparing the output from the *what* command with the *slist*.

2.3 MAKE - Minimal Work Facility

Make is a tool for building programs [SIF]. Information on the inter-file dependencies and the commands necessary to build a process are stored in a *makefile*, and executed repeatedly and reliably. In addition to the information contained in the *makefile*, and transparent to the user, *make* also uses the time of last modification of every file referenced in the *makefile*. This attribute is maintained by the UNIX file system, and is updated every time a file is modified. Figure

3. The hierarchy of *slists* is a natural organization, since the UNIX file system is a rooted tree with an arbitrary number of levels.

4 shows the function of *make*.

Make builds a product by assuring that its components are up to date and rebuilding only what is necessary. It avoids wasteful recompilation, and inclusion of out of date components.

Make allows a third party to manufacture software in a consistent manner, by repeating the building commands stored in the *makefile*.

The use of *make* has been generalized to:

build the directory structure for constructing a product.
extract the source files from the SCCS archives.
perform checks on the source files.
build the product.
perform the unit tests on the product.
install the product in an appropriate place.
produce documentation or listings.
generate *slists*, verify consistency, and stamp products.

Further, *makefiles* can be arranged in a hierarchy paralleling the *slist* hierarchy. The *makefiles* themselves become SCCS files, and every *slist* must reference at least one *makefile*. The lowest level *makefiles* build a single process; the *master makefile* builds an entire system.

Since *make* constructs a graph of dependencies, and tests each to see if anything needs to be rebuilt, it requires time to determine that nothing needs to be done. For a complex system with relatively short component build times, the overhead has been measured at approx. 20% of the time to rebuild from scratch. In more typical cases the overhead is much less.

In any case, the machine time is well spent, since using *make* a large class of inconsistency errors can not possibly occur.

3. SOFTWARE MANUFACTURING

The Manufacturing System is based on all the concepts described so far. The manufacturing process can be subdivided into three major parts: software turnover, consistency checks, and the building proper.

Software turnover from development to manufacturing occurs when the developers install all source files in the SCCS archives, and deliver to manufacturing the *labels* of the *slists* for the process, subsystem, or system. Next, a series of automatic and manual checks are performed on the source files and the *slists*. Finally, the process is built, tested, and installed.

Figure 5 shows how the manufacturing system interfaces with the development and manufacturing environments.

3.1 Software Turnover

To build a *system*, the following inputs are required:

a. A schedule with the dates at which the various parts of the system will be available. A system is built in layers, as follows:

the shared data, e.g. common files, libraries, etc.
the basic tools: operating systems, compilers, loaders, and the system building tools themselves.
several layers of application processes.

As soon as one layer is successfully built, the system is released to the user community, to serve as a base for building and testing the next layer.

b. Process information, i.e. the *label* of the process *slists*. Besides the *slist* information, the developers also submit a copy of the process, which is checked for consistency with the *slist*, and if consistent, added to the system as is.

c. In the majority of cases the *label* of the *master slist* of the previous system is also required, to identify the processes that changed since it was released.

The turnover itself takes place when the developers submit the *labels* of the respective *slists* according to the schedule. This information and the compliance with the schedules are checked by the manufacturing system, and the approved *slists labels* are used to update the hierarchy of indirect *slists* all the way to the *master slist*. When the required set is complete, the consistency checks are started.

3.2 Consistency Checks

A *system* is defined to be a collection of consistent processes. For processes to be *consistent*, they must pass all the tests described below.

The software manufacturing system performs various consistency checks on the software, prior to, and during the building phase. These checks are:

a. Hierarchy check: the entire collection of *slist*s and the *makefile*s required for the system must be present, i.e. there may be no missing *slist*s.

b. Global consistency check: all files referenced more than once (in one or more *slist*s), must be referenced with the same *label*.

c. Shared files check: files referenced in more than one *slist*s must reside in a common area. It is illegal to have two (or more) copies of the same file in the system.

d. Processes must be stamped with the SCCS id of the respective *slist*, and their constituent parts must agree with the *slist*.

All the checks described are automatically performed by the manufacturing system. If any fail, the *slist* is returned to the developer, and it must be resubmitted after correction.

3.3 Building a System

The software manufacturing system consists of several Shell procedures [SRB], that invoke *SCCS* and *make*.

The system is built by issuing a single command, regardless of how much of the system has to be built. The underlying tools will perform the minimum amount of work required to build, or change the system. Specifically, the following command is executed to start the manufacturing process:

```
mksys release3.0 3.17
```

The above example directs the manufacturing system to build the system version 3.0, at the version 3.17 of the *master slist*.

The manufacturing process will perform the following steps, in addition to the consistency checks that were previously described:

a. Will get the *master slist* from the SCCS archives at the version required, e.g. 3.17.

b. Will get the *master makefile* from the SCCS archives, at the version specified in the *master slist*.

c. Will invoke the *make* command, using the *master makefile* as input. Beginning at this point, the entire building process consists of walking the hierarchy of *slist*s and *makefile*s, until the system is built. Every *makefile* executes the following steps:

   1. Create the directory structure that the respective process will require.
   2. Get all the source files required for the process from the SCCS archives. Each file is extracted at the version specified in the associated *slist*.
   3. Check the files to insure compliance with project standards.
   4. Build the process.
   5. Perform the unit test on the process.
   6. Check the source file needed for documentation.
   7. Build the documentation for the process, e.g. running instructions, specifications, etc.
   8. Verify that the process and the *slist* are consistent.
   9. Stamp the process with the *label* of the *slist*.
   10. Install the process and the associated documentation in the appropriate directories.

   The hierarchy of *makefile*s will insure that work already performed is not repeated, thus providing an incremental building capability.

The contents of a system are controlled by the contents of the *slist* and *makefile* hierarchy, i.e. only the processes whose *slist*s appear in the master will be built, and only the commands appearing in the *makefile*s will be executed.

4. CRITIQUE AND FUTURE WORK

The Software Manufacturing System based on a hierarchy of *slists* and *makefiles* has performed well. Our experience has indicated that improvements can be made in several areas.

4.1 Shared Data

The shared data, i.e. global header files[4], libraries, etc., introduced the problem of distributed ownership on a system resource.

Since there were numerous owners for the shared files, the shared data started showing a scattering tendency, as well as duplication of global data.

The solution for this problem was to implement the following:

a. The shared data (among other things) are stored in a project database called the System Glossary [PAB].

   The Glossary helps identify redundant or inconsistent data, and contributes to the organization of the shared data. The contents of the Glossary can also be used to generate the shared files in a given programming language.

b. Shared files reside in relatively few, well known directories, e.g.:

   shared header files
   shared libraries
   shared data files

As a fringe benefit of this organization, the *makefiles* can now be generated consistently, since all references to shared data are to the places mentioned above. In fact, a substantial part of a *makefile* can now be produced by a specialized tool.

4.2 Structure of SCCS Archives

In the first version of the Software Manufacturing System, the directory structure of the manufactured system mirrored that of the SCCS archives. The knowledge of the structure was embedded in the *makefiles*. As a result of this tight structural coupling, neither structure could evolve, except by addition.

We first considered a scheme permitting multiple logical views of a single physical structure (the SCCS structure), after which, we settled on:

a. The structure of the SCCS archives is no longer publicly known. The files are moved to and from the SCCS structure entirely based on the name of the file (which are unique).

b. A *makefile* no longer knows about the SCCS structure, the only structural knowledge allowed in a *makefile* is:

   current directory[5], and possibly a tree whose root is the current directory (this structure will be built by the *makefile* itself)

   the well known directories for the shared data

The decoupling of the SCCS and system structures allows the SCCS structure to evolve without creating havoc with system-wide effects.

This also implies that a *makefile* will build a process the same way, regardless of its location. Therefore, the same *makefile* serves the developer - in a private work space, and the manufacturing system - in the official system space.

4.3 Differential Building Tools

The building of a large software system always poses the problem of sheer size. It takes a long time and considerable machine resources to build it. In addition, during system test, there is the need to make "quick fixes" as the testing progresses.

Since no patching of the object is allowed (i.e. all the changes must be made at the source code level), it is necessary to rebuild only the affected part of the system.

The solution entails differential building tools, e.g. tools that rebuild only the subsystems that have changed, while still maintaining a consistent system. The differential building tools do the following:

a. Impact assessment: using the *slist* hierarchy as a Table of Contents for the entire system, all the objects affected by a proposed change can be identified.

4. A header file is a source file containing global data; it is included by other source files.

5. The UNIX system supports the notion of a current directory: the directory from which it begins the search for file names.

b. Change propagation: a change in a source file will typically trigger changes in the *slist* referencing the file, as well as in all the indirect *slists* all the way to the *master slist*.

c. Change confirmation: a change in a shared file may not affect a source file that includes it, if the source file does not use all the symbols in the shared file.

The differential building tools minimize the work to be done even further than the primary software manufacturing tools. The differential manufacturing method has the potential to reduce the time and resources required to build a system by approximately one order of magnitude.

4.4 Multimachine Environment

The problem of distributed computing, in all its aspects, is something yet to be solved. As far as software manufacturing is concerned, several problems are introduced by having a multimachine environment, even with all the machines under the same operating system. The problems we found were:

Parts of the SCCS archives had to reside simultaneously on several machines, triggering a need for a distribution system to insure consistency and integrity of the files.

The name uniqueness requirement became harder to enforce, since each SCCS structure needed to produce a list of names to be shipped to, and merged on one machine, where the enforcement checks were performed.

It was necessary to designate for each source file the machine on which the changes are allowed, thus increasing the complexity of the SCCS tools.

The ideal solution would be to have an operating system supporting the "virtual machine" concept, e.g. the user does not know or care what the distributed environment looks like.

The actual solution was to permit read-only access to other UNIX machines [ALG2], and to develop a higher level of SCCS commands that will execute on a remote machine the same way they execute on a local machine. This arrangement requires only one copy of the SCCS archives (on one machine), thus eliminating the consistency and integrity problems.

5. ACKNOWLEDGEMENT

The authors wish to acknowledge the contributions of C. B. Hergengan, G. B. Garst, A. L. Glasser, and D. W. Ting who built many of the tools the manufacturing system depends on. They also wish to thank K. G. Freeman for his direction, support, and helpful criticism.

REFERENCES

[ALG] Glasser, A. L., "The Evolution of a Source Code Control System", Proc. Software Quality and Assurance Workshop, Software Eng. Notes, Vol. 3, No. 5, November 1978, pp 122-125.

[ALG1] Glasser, A. L., "A Simple Source Control System", unpublished work, 1980.

[ALG2] Glasser, A. L. and D. M. Ungar, "A Distributed UNIX System", paper submitted to COMPSAC-80.

[BWK] Kernighan, B. W. and D. M. Ritchie, "The C Programming Language", Prentice-Hall, 1978.

[DMR] Ritchie, D. M. and K. Thompson, "The UNIX Time-Sharing System", The Bell System Technical Journal, July-August 1978, Vol. 57, No. 6, Part 2, pp 1905-1930.

[MJR] Rochkind, M. J., "The Source Code Control System", IEEE Trans. on Software Engineering, Vol. SE-1, December 1975, pp 364-370.

[PAB] Blosser, P. A. and B. C. Wonsiewicz, "The System Glossary: A PSA Based Process And Data Dictionary", unpublished work, 1980.

[SIF] Feldman, S. I., "Make - A Program for Maintaining Computer Programs", Bell Laboratories Computer Science Technical Report No. 57, 1977.

[SRB] Bourne, S. R., "The UNIX Shell", The Bell System Technical Journal, July-August 1978, Vol. 57, No. 6, Part 2, pp. 1971-1990.

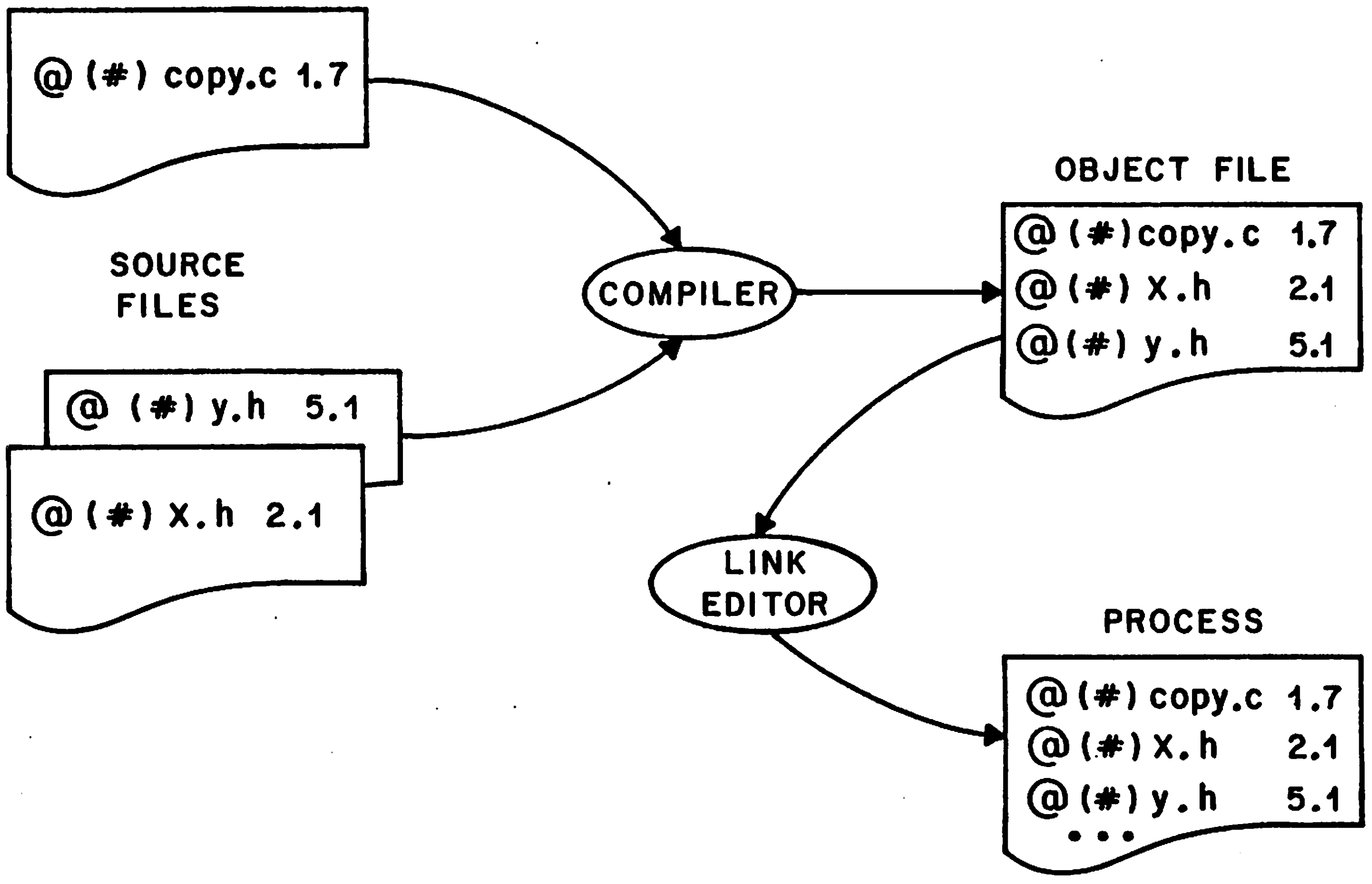

FIGURE 1. LABEL PRESENCE IN THE SOFTWARE

GC 4/30/80

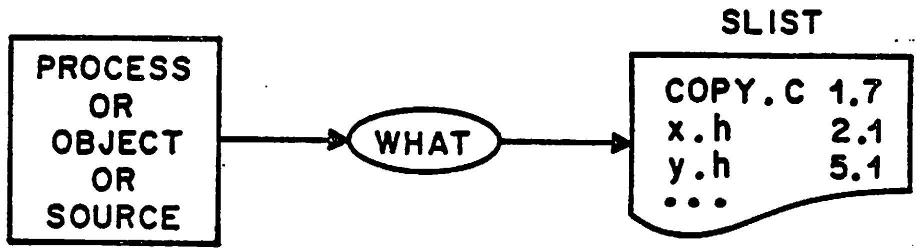

FIGURE 2. THE "WHAT" COMMAND

GC 4/30/80

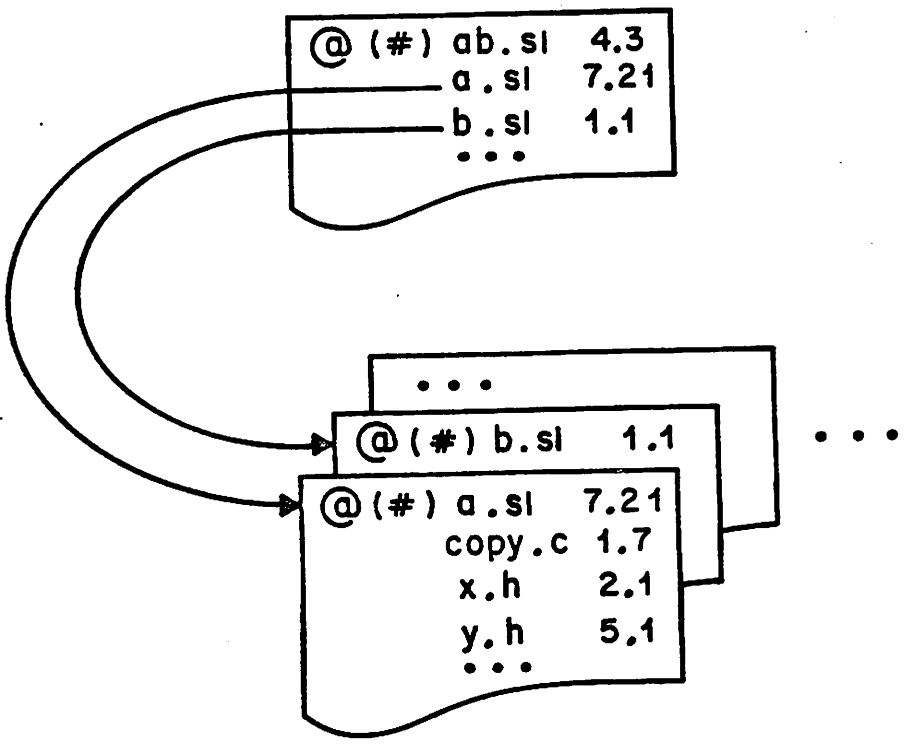

FIGURE 3. HIERARCHY OF SLISTS

GC 4/30/80

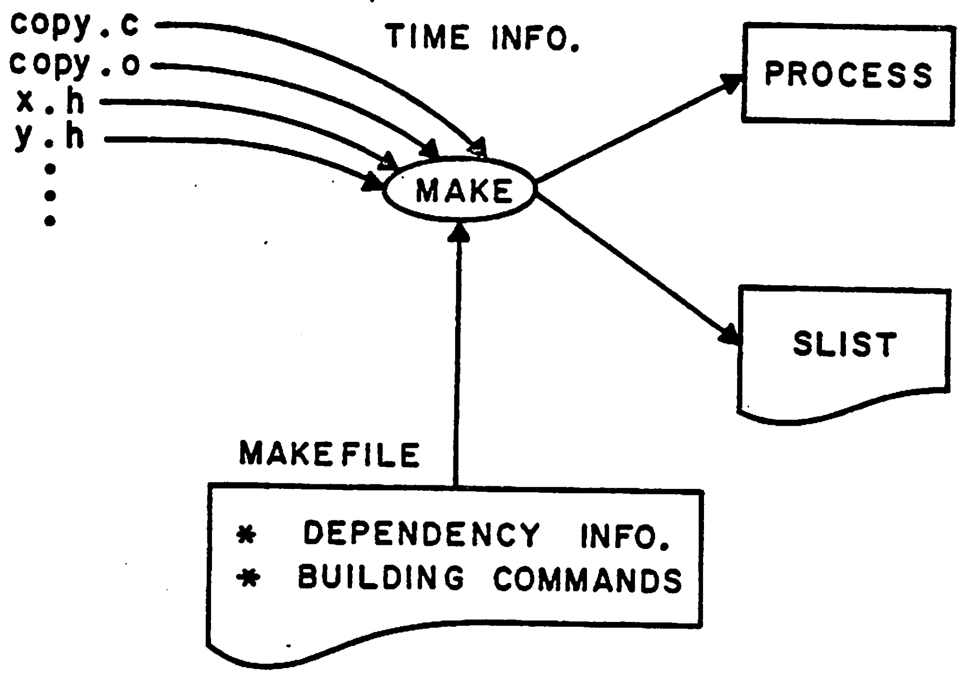

FIGURE 4. THE "MAKE" COMMAND

GC 4/30/80

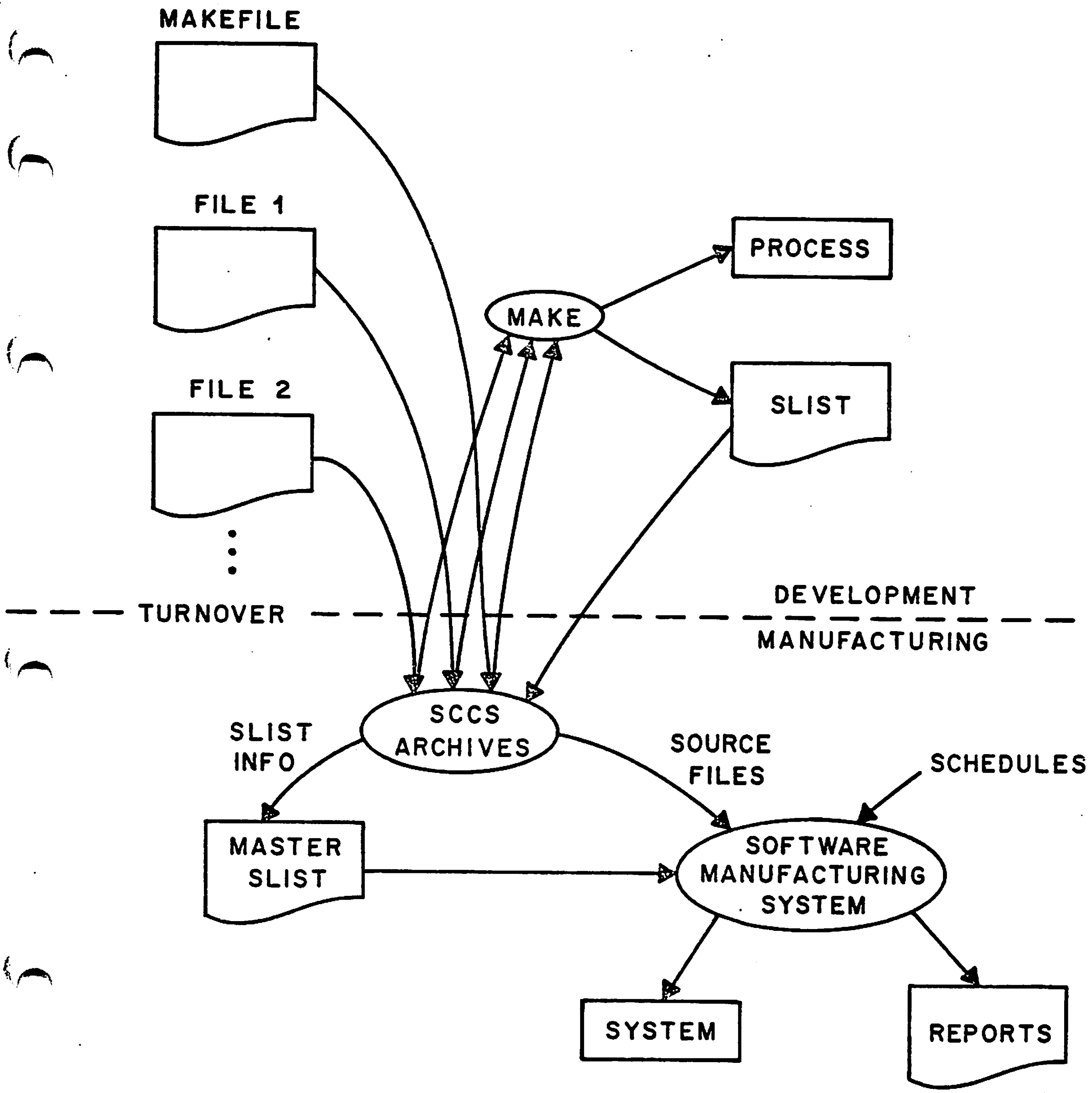

FIGURE 5. THE SOFTWARE MANUFACTURING SYSTEM

GC 4/30/80