SUN C TO SUN ANSI C CONVERTER

by

Laura L. Agee

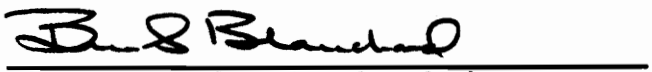
Project and Report submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

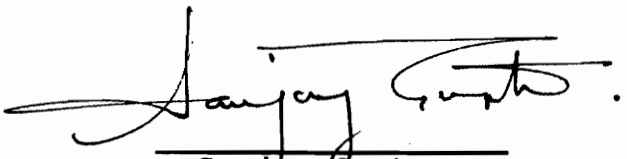
in

SYSTEMS ENGINEERING

APPROVED:



B.S. Blanchard, Chairman



Sanjay Gupta

C.M. Waldorfer

May 1995

Blacksburg, Virginia

Sun C to ANSI C Converter

by

Laura L. Agee

Committee Chairman: B. S. Blanchard
Systems Engineering

(ABSTRACT)

Future operating system vendor support for an office building LAN was investigated in response to the realization that current vendor operating system support of the LAN was going away. Options analyzed that would best preserve the existing functionality of the software applications utilized on the LAN were, (1) to do nothing, (2) utilize another vendor for support and (3) upgrade to the current vendors latest operating system, Solaris 2.2.

The feasible decision was to upgrade to Solaris 2.2. It offers a software system design that can be implemented to preserve existing functionality of the LAN's software applications. Specifically, the software design resolves incompatibility differences between existing K & R C code and the ANSI C compiler that is to be utilized under Solaris 2.2.

A representative approach was taken in the software system design by utilizing Data Flow Diagrams. Also included in the design are the screen displays needed to interface with the user. A database is implemented into the system design to aid in the testing of performance requirements.

Table of Contents

List of Figures

Page

List of Tables

1.0 Identification Of Need

1.1 History

The Martin Marietta Data Center (MMDC) is a five story office building that contains approximately 600 Martin Marietta employees. There are thirty six Sparc/2 workstations and two hundred and seventy five Sparc ELC workstations distributed throughout the building. All of the workstations are connected to either 4/75, 4/670 or 4/690 file servers. The total number of these file servers is thirty nine and they are all located in one room on the third floor. Each file server supports up to eight workstations. These workstations and file servers make up the Local Area Network (LAN) within the MMDC facility. Sun Microsystems is the manufacturer/ vendor for all of the above mentioned hardware. They also supply the Martin Marietta Data Center with the operating system software that is to run on the LAN hardware. The operating system that Sun Microsystems provides is a Unix variant and is referred to by Sun Microsystems as SunOS. The LAN does not extend outside the bounds of the five story building.

1.2 Functional Requirements

Six "custom made" applications have been developed that are front end interfaces to a database (Sybase) and contain a total of 530,000 lines of Sun C code, developed in open

windows. The following two paragraphs explain the facilities need for two of the six "custom made" applications. The other four, not elaborated on, support similar types of functions.

A "custom made" application has been developed that allows the logistics organization to perform the following required functional tasks.

1. Ability to perform "on the spot" audits of both hardware and software within the facility on any given day Monday through Friday from 7:00 A.M. through 4:00 P.M.
2. Record the up to date location, history and purchase order of each work station, file server, disk drive and software within the facility.
3. Provide the customer with daily reports of what hardware and software items have been shipped out for repair and when they are due to return.

A second "custom made" application was developed to allow security personnel to perform the following required tasks.

1. Ability to promptly retrieve and view current security information on any employee upon customer request.
2. Process up to 600 visitor requests a day.
3. Provide the customer with daily visitor request reports. Each report must include the visitors clearance status, arrival time, departure time and

weather or not an escort for the visitor is needed. The "custom made" application also satisfies the following required functional user need.

4. Each employee within the facility must be able to simultaneously send visitor requests to security via soft copy.

In summary, the above "custom made" application satisfies the contractual agreement to ensure that non cleared personnel only enter the facility when escorted.

An office automation COTS package, supplied by APPLIXware, is accessible to all users within the facility to satisfy the following required functional tasks.

1. All users must have word processing capability needed for the development of documents.
2. All users must have the ability to create schedules and multi color charts for presentation purposes.
3. Each employee must have the capability to send mail, via softcopy, to any other employee within the facility.

1.3 Identification of Deficiency

The functionality of the "custom made" applications will no longer be supported due to the lack of operating system support by Sun Microsystems. Sun Microsystems will not be supporting SunOS 4.1.3 after July 1996. Solaris 2.2, also

known as SunOS 5.2, is an upgraded version of SunOS 4.1.3 and is fully supported by Sun Microsystems. The MMDC has a contractual agreement with the customer that vendor support for the operating system on the LAN must exist. A feasible solution needs to be made that will resolve the inadequate operating system support and at the same time, preserve all the functionality of the functional tasks discussed in section 1.2.

2.0 Feasibility Analysis

Due to the loss of vendor support of the existing operating system on the LAN, three possible alternatives need to be investigated that will best preserve the current functionality of the "custom made" applications. The alternatives are to (1) do nothing and continue on using the current operating system (2) Upgrade to Solaris 2.2 and (3) utilize a new operating system on the LAN.

2.1 Use Current Operating System

The "custom made" applications on the LAN were developed, in part, by using Sun supplied "proprietary" libraries that were not purchased, but licensed with the annual operating system maintenance contract. These libraries will not be licensed, and thus not available for use, once support of SunOS 4.1.3 goes away. These libraries are, however, still supported by Sun Microsystems under the Solaris 2.2 operating system. Without these libraries the existing "custom made" applications developed on the MMDC LAN will cease to operate and the required functional needs, discussed in section 1.2, of the facility will not be met. Thus, To stay with the current operating system is not feasible.

2.2 Utilize a New Operating System

Operating systems are made for specific hardware platforms. The kernel is what controls the operating system and tells it how to communicate with hardware i/o ports. Each

vendor provides their customers with an operating system that is compatible only with their specific hardware platform. Sun Microsystems operating system is compatible only with a SPARC-based hardware platform. In summary, their is no generic operating system that runs on all unix based hardware platforms. To migrate to another vendor for operating system software support would not be possible since the current hardware platform within the facility would not be compatible with another vendors operating system kernel. Thus to migrate would require not only an operating system software migration but also a hardware platform migration.

2.3 Upgrade to Solaris 2.2

Determination of the impacts that the operating system upgrade will have on the LAN are discussed in the following sections.

2.3.1 Impact on LAN Hardware

As mentioned earlier, the vendor/manufacturer of the current hardware platform that the MMDC facility utilizes is Sun Microsystems. Sun's upgrade to Solaris 2.2 did not create any incompatibilities with their SPARC-based hardware platform. Thus, there is no impact on the LAN hardware due to an upgrade to Solaris 2.2.

2.3.2 Impact on "Custom Made" Applications

The current compiler on the MMDC LAN only supports Sun C code, equivalent to Kernigan & Richie C code. Thus, the

"custom made" applications were all written in Sun C. The Solaris 2.2 operating system will not support a Sun C compiler. Therefore, the operating system will not support applications developed in Sun C code. The compiler under Solaris will only compile the industry standard C code known as Sun ANSI C. Because of this, all the "custom made" applications will need to be converted to ANSI C and recompiled under the Sun ANSI C compiler. This will allow the application executables to be run under the Solaris operating system. According to Sun Microsystems, the Sun ANSI C compiler is 95% compatible with Sun C code syntax. Sun Microsystems determined this by evaluating representative Sun C source code at their development facility. A list of 20 incompatibility differences between Sun C code and Sun ANSI C code have been provided to the MMDC by Sun. Each line of Sun C code within all six "custom made" applications will need to be evaluated against the list provided by Sun Microsystems to determine if an incompatibility exists. If so, each incompatibility detected will need to be corrected with the appropriate ANSI C syntax.

2.3.3 Impact on Office Automation Software Package

The office automation package (discussed in section 1.2) was developed and delivered by a third party vendor, APPLIXware, as compiled executables. The executables that APPLIXware has provided the MMDC is compatible with the

Solaris 2.2 operating system.

2.3.4 Conclusion

From the impacts identified above a system that converts "custom made" applications over to Sun ANSI C syntax needs to be developed. Otherwise the "custom made" application executables will not run on the LAN. The MMDC management has tasked the software engineering group to determine a feasible conversion approach to resolve the problem. In section 2.0 a feasible conversion approach is determined. Existing hardware and software within the MMDC is to be utilized where ever possible to aid in any development work that may be needed to resolve the conversion problem. The concerns addressed above must be resolved by July 15 1996 since this is when support ends for the SunOS operating system.

3.0 Feasible Conversion Approach

The goal is to make existing Sun C code compatible with the Sun ANSI C compiler, to ultimately allow Sun C executables to run, error free, on the Solaris operating system. The approach that assures the least amount of time and is the most accurate will be the most cost effective approach to take. In determining what the best approach should be to resolve the problem previously stated, pertinent aspects of each approach being considered needs to be analyzed.

3.1 Use An Existing Commercial Product

The major "difference" that Sun C code has with ANSI C is how functions are prototyped. Note the emphasis placed on difference. ANSI C's prototyping technique may be a difference but not an incompatibility. Although ANSI C provides this new style of prototyping, ANSI compilers still fully support code written in the old prototyping style. Thus the prototyping difference between Sun C and ANSI C is not a concern of "custom made" application conversion. The concern of this conversion is to correct the differences that are incompatible (will cause existing applications to fail when compiled under the ANSI compiler). Thus, the 20 incompatibility types that Sun Microsystems has provided do not include prototyping differences.

A few programs currently exist that address C to ANSI C differences. All these programs were designed for specific

compilers and are prototype generators. The programs convert regular C prototyped function definitions and declarations to the new ANSI C style. One in particular, known as *Protoize*, was a unique patch for the FSF GNU C compiler but did not handle full blown translation from C to ANSI C. The program is a filter which sits between the preprocessor and the next compiler pass and converts old style prototypes to new ANSI C style. It was not made to be a generic converter program. No current program exists that addresses the syntax oriented types of differences that are the true incompatibilities. It appears that organizations are left to come up with their own methods of resolving the incompatibility problems that exist between regular C and ANSI C.

3.2 Use a Manual Approach

The next approach looked at is to traverse through each line of the existing Sun C code and make corrections wherever needed. Section 1.2 stated that there are 530,000 lines of Sun C code that comprise the six "custom made" applications. From section 1.3.2 it is determined that each line of Sun C code will need to be looked at and compared to the list of 20 possible incompatibilities types to determine if an incompatibility does exist. Some assumptions will need to be made in determining how long something like this would take using a manual approach.

1. Upon review of the 20 possible incompatibilities

types that exist (discussed in detail in the design section), it is estimated that it will take an average of 5 minutes to correct an incompatibility type once it is detected.

2. If the lines of code are manually traversed it is estimated that each line will take an average of one minute to determine if an incompatibility exists.
3. For the most part, each line of Sun C code within each of the "custom made" applications contains single event C commands or declarations. Thus, it is assumed, that on average only one incompatibility would exist on a single line of code.
4. Since 95% of existing source code is compatible with the Sun ANSI C compiler (from section 1.3.2) it is assumed that only one incompatibility will be detected per every 20 lines of code. Thus, an estimate total of

   530,000(lines)/20 = 26,500 (Eq. 1)

   incompatibilities could currently exist within Sun C code and would need to be detected and corrected in order for the application executables to run on the LAN.

Based on the above stated assumptions, an estimate of how long it would take to convert the six "custom made" applications is determined. It would take 530,000 minutes (4.24 yrs) to

traverse each line of code and

26,500*(5min)*(1hrs/60min) = 2208 hrs (Eq. 2)

or 1.06 yrs for one person to correct all the incompatibilities that exist. A total of 2.65 years utilizing 2 software engineers. This yields an estimated cost of $193,500. This time assumes no rework due to inconsistent detection & correction due to fatigue of which humans are subject to.

3.3 Use an Automated Interactive System

Each one of the 20 incompatibilities (described in detail in the design section) that have been provided to the MMDC were reviewed by software engineering to determine if automating their detection and correction is possible. It was determined that detecting all of the 20 possible incompatibility types within Sun C code files could be automated. From the list, 10 of the 20 would require inductive reasoning in order to be corrected, thus correction of these types would have to be done manually. The other 10 were of a deductive nature and correction for them could be automated. Thus a fully automated system is not possible. The system would have to be somewhat interactive.

3.3.1 Development Time

ROM's, rough order of magnitude estimates, were provided from the software engineering group on the estimated development time it would take to design and code the

"automated interactive system". It is estimated to take two weeks per incompatibility for two software engineers to design and code the system. This is a total of 40 weeks or approximately 10 months. An estimated cost of $56,000.

3.3.2 Test Time

Given the fact that there ar 6 applications that will need to be tested, the test group provided a one person ROM of 1 week of testing per application. An estimated salary cost of $4,000.

3.3.3 Operational Conversion Time

Since half of the incompatibility types that exist are to be resolved through inductive reasoning and thus need human intervention to resolve them, it is estimated to take

26,500lines/2 X 5min/1hr X 1hr/60min X 1day/8hr X 1week/5days = 27 weeks (Eq. 3)

or .53 years for 1 software engineers to correct these incompatibilities once the system is available for use, and only .265 years, or 14 weeks, for 2 people. An estimated salary cost of $19,600. This 14 week span does not account for time lost due to any hardware or software failures that may occur when the system is in operation (sections 3.4.1 and 3.4.2 account for this time). The "automated interactive system" will be responsible for detecting all incompatibilities as well as correcting those that do not require inductive reasoning. The time for the system to do

these last two tasks is negligible in comparison to the manual approach. Figure 1. Below summarizes the activities involved in the interactive automated approach.

ACTIVITY PHASES	TIME	RESOURCES	COST($)
Design	40 Weeks	2 S.E.	56,000
Test	6 Weeks	1 T.E.	4,000
Operation	16 Weeks	2 S.E.	19,600
Total	**60 Weeks**		**79,600**

Figure 1. Life Cycle Phases.

3.4 Conclusion

From the analysis discussed, the "automated interactive system" approach saves the most time and thus is the most cost effective approach, that is if it converts accurately. In section 3.4.1 below, the accuracy that the system must have is specified. The automated system will take one and a half years less time, to convert the applications, than the manual approach. Most importantly it will allow for a solution of the conversion problem prior to the need date of July 15, 1996. If development of the system begins in mid April of 1995 all code could be converted by mid June 1996. This leaves a 4 week buffer period between estimated completion date and actual "need" date desired for completion. This

implies that the system may have up to 4 weeks to correct any hardware or software failures that may arise (all failure types are explained in detail in the maintenance concept section).

3.4.1 Software Specifics

The software components needed for system operation are:

1. The SunOS operating system
2. Sybase database to make log entries for test purposes (as specified in the requirements section that follows)
3. The "automated interactive system" developed software.

The 4 week buffer period, discussed in section 2.4, is used to determine a quantitative accuracy value, in detection and correction, of the system. Once testing is complete, it is estimated that it should take an average of 6 minutes to make a correction to the "automated interactive system" if a failure occurs. A failure either being a compile time error or a run time error detected in the "custom made" application after it has been converted. If the automated system is 98% accurate then

$$26{,}500 \times .02 = 530 \qquad \text{(Eq. 4)}$$

incompatibilities out of the 26,500 that are expected to exist, within Sun C code, will not be detected/corrected properly. If each of the 530 failures take an average 6

minutes to correct, it will take .66 weeks time for two software engineers to make corrections to the "automated interactive system" due to detection failures. This time frame falls well within the 4 week buffer time. As noted above, the 6 minute time frame is software corrective maintenance time during system operation. It is assumed that the six week test period will have resolved most of the failures that would have required lengthy corrective maintenance rework. Thus, 6 minutes is realistic for software corrective maintenance time.

3.4.2 Hardware Specifics

The hardware components needed to utilize the automated system are listed below:

1. 2 Sparc-2 workstations (one for each software engineer)
2. A SCSI disk drive
3. A printer to print out test information

Since the automated system will be in operational use from March until the end of June (1996), the system will not need to account for any preventive maintenance actions. Mean Corrective Maintenance time (Mct) for any one of the three hardware components listed above will take 30 minutes. The Corrective maintenance time accounts for how long it would take hardware maintenance to swap out the hardware component and replace it with a new one. The Maintenance Concept

section will fully describes all hardware and software maintenance tasks.

Due to the short duration of the operational life cycle, it is only expected that a maximum of one failure for each hardware component is likely to occur during operational use. Thus, a MTBF value of 320 is expected for all hardware components.

The corrective maintenance times for both hardware and software and the raw conversion time of 14 weeks, to convert the "custom made" applications to ANSI C, determines operational life span of the system. The operational life cycle shall be no longer than 16 weeks. From this, it is determined that the system will need to traverse through

530,000/16*1/5 = 6625 (Eq. 5)

lines of code per each eight hour work day, five days a week, from April through June of 1995. From the analysis above the availability of the system should be approximately 95%. This estimate is based on the operational span of 16 weeks as compared to hardware and software corrective maintenance times.

4.0 System Operational Requirements

4.1 Mission Definition:

4.1.1 Automate a system that will convert "custom made" applications within MMDC from Sun C over to Sun ANSI C. There is a total of 530,000 lines of code to convert.

4.1.2 The system is to detect all incompatibilities.

4.1.3 The system will only correct incompatibilities that do not require inductive reasoning.

4.1.4 The automated system is to be interactive to allow software engineer intervention when inductive reasoning is needed.

4.1.6 Make use of existing hardware and software within the MMDC to automate the system.

4.1.7 A database must be used to uniquely log every incompatibility detected/corrected by the system and the total number of incompatibilities found. This will provide a mechanism for the test group to determine the accuracy of the system. This also documents what code changes were made to the "custom made" applications during conversion.

4.1.8 The operational profile is for the system to successfully operate five days a week Monday through Friday.

4.2 Performance and Physical Parameters:

4.2.1 The system must have an accuracy rate in detection and correction of 98% (explanation found in 3.4.1). That is 98% accurate in detection and 98% accurate in correction.

4.2.2 6625 lines of code must be converted per 8 hour work day in order for conversion to be completed by the June 1996 time frame requirement specified in the usage requirements section.

4.2.3 The capacity of the database must allow for 26,500 entries into the database. This is assuming that one incompatibility would be detected per 20 lines of Sun C code with a total of 530,000 lines of code in all.

4.3 Use Requirements:

4.3.1 The automated system will be utilized by the software engineering group.

4.3.2 The eight hour usage span will fall in line with the working hours of the software engineering group. This is any where from 7:00 AM to 4:00 PM Monday through Friday.

4.3.3 The months of operation will fall between March 15 1996 through the end of June 1996. The operational profile for these months is shown in Figure 2. on the following page.

4.3.4 The automated process shall be designed to convert

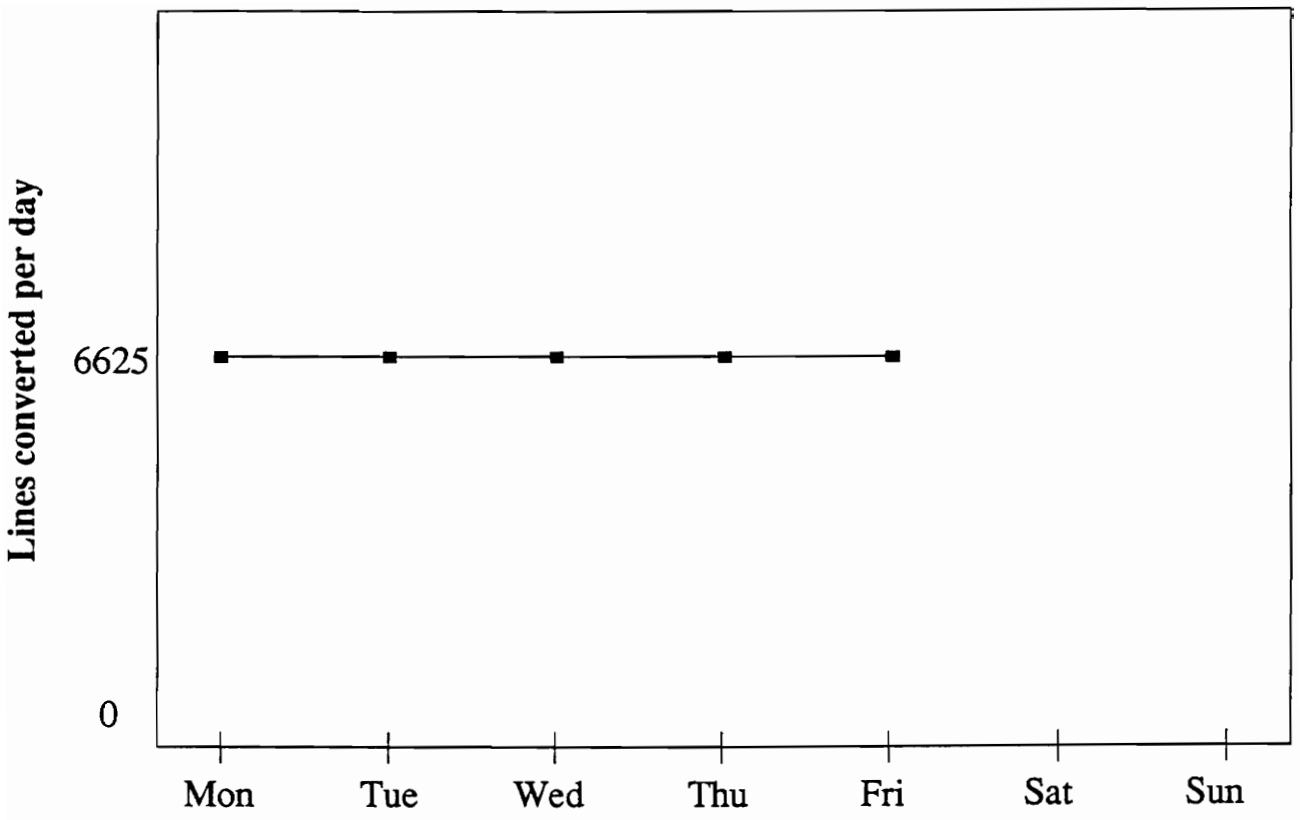


Figure 2. System Utilization Profile (March 1996 to June 1996)

an entire "custom made" application at a time.

4.3.5 Once the automated process has traversed through 6625 lines of code, the system must log the time/date of detection, the line count and the elapsed time it took to reach a line count of 6625 lines of code.

4.3.6 The option to print out a performance status sheet must be provided by the automated system. Information displayed on the sheet must allow the software engineer to determine if 6625 lines of code are being traversed by the system in the course of an eight hour day. The sheet must also provide incompatibility detection totals to aid the software engineer test personnel in determining the over all accuracy of the system.

4.3.7 Information about detected inductive incompatibility types will be displayed to the screen to allow the software engineer to locate the incompatibility within a file. The information displayed to the screen will be the source file name and the line number within that file where the incompatibility resides.

4.3.8 The automated system shall provide options for the user to compile and run an application once it has been converted.

4.4 Operational Deployment or Distribution

4.4.1 The number of personnel that are to utilize the system are two software engineers.

4.4.2 The system is to run on 2 SPARC workstations under Sun Microsystems version 4.1.3 operating system that resides within the MMDC.

4.4.3 Database storage will reside on one file server disk drive. One printer must be provided to satisfy the print requirement stated in the use requirements section. Both the disk drive and the printer will reside within the MMDC.

4.4.4 The automated process is to be used solely to convert the currently utilized applications (530,000 lines of Sun C code) within the Martin Marietta Data Center from Sun C to ANSI C.

4.4.5 The source code and executable of the developed system are to reside in software engineering's source and application directories.

4.4.6 A Sun ANSI C compiler will need to be delivered for software development testing use.

4.4.7 The automated system is to be ready for use by mid April 1996.

4.5 Operational Life Cycle:

4.5.1 Thus the operational life cycle span, or conversion time, of the automated system shall be no longer

than 16 weeks.

4.5.2 The personnel that will operate the system is the software engineering group.

4.6 Effectiveness Factors

4.6.1 The skill levels needed for operational use of the system are software engineers with a minimum Level 2 status.

4.6.2 Operational tasks include compiler knowledge, utilizing database query language, and rewrites of Sun C code for corrective maintenance.

4.6.3 The system is to be available for use 95% of the time.

4.6.4 The MTBF of the converter software shall be 2.2 hours for software failures and MTBF for hardware failures shall be 106 hrs.

4.6.5 The Mct of the converter software shall not exceed 6 minutes. The Mct shall be 30 minutes for hardware failures.

4.7 Environment:

4.7.1 The temperature surrounding the SPARC workstations stays at approx 65 to 72 degrees fahrenheit.

4.7.2 The equipment already exists within the building and thus will not be exposed to external air temperatures.

4.7.3 The humidity within the building is maintained at 40

to 60 % relative humidity.

5.0 Maintenance Concept

5.1 Levels of Maintenance

The anticipated levels of maintenance that are pertinent during system operation are known as Organizational, Intermediate and Producer maintenance. Mean Corrective Maintenance (Mct) will be performed by HMO and Sun Microsystems for hardware failures and software engineering and Sybase for software failures.

5.1.1 Organizational

Tasks at this level will involve regular database dumps of the database and running the database consistency checker (DBCC) against the database. This is a Sybase command that detects and corrects problems with data within a database. These tasks will be performed by software engineering. Neither of the two above mentioned tasks require down time for the database. Thus, these tasks can go on while the system is in operational use. If a problem is detected using the DBCC for the database, down time will then be needed to make corrections to corrupt data. Finally, visual inspections of the automated process will occur upon use of the system to make sure that it appears to be operating correctly.

5.1.2 Intermediate

Intermediate maintenance occurs when actual fixes to the system need to be made. The database maintenance at this level will be provided by software engineering. Upon failure,

either the Sybase **dbrepair** command or the Sybase **checkdb** command, with the fix option, will need to be issued. If the database integrity is not restored Sybase will need to contacted to provide guidance for the fix. Some other software engineering maintenance tasks will include restoring corrupt data that has been logged to the database, which entails reloading backups from database dumps, and making code corrections to the automated process to correct minor problems that arise from visual inspection.

HMO disk drive maintenance, at this level, will be to first try rebooting the disk drive upon failure. If the disk drive has failed due to a corrupt sector, HMO will swap out the disk drive and replace it with a new one. The database will then be reloaded onto the hard drive by software engineering. The workstation hardware components (monitor, CPU, etc...) will also be swapped out by HMO if they should fail.

5.1.3 Producer:

Maintenance tasks at this level will come from software engineering for the developed converter software, Sun Microsystems for hardware and operating system software failures, and finally Sybase for database failures that can not be resolved with sybase repair commands. If any operating system incompatibilities with the database (SYBASE) arise, Sun Microsystems will need to be contacted to send patches for the

operating system fix. Software engineering producer maintenance will include such tasks as major code rewrites to correct failures in detection and correction of the automated system that are revealed during the usage and test phase of the systems life cycle. Disk drives that are swapped out due to failures will be to shipped to Sun Microsystems for repair.

5.2 Elements of Logistic Support

The major elements of logistic support for the system will be the utilization of the Sun ANSI C compiler and the software engineering groups manual resources in code repair. The logistics organization within the MMDC will provide spares for failed hardware utilized by the system.

5.3 Repair Policies

After an application is converted, the application will then be compiled under the ANSI C compiler to determine the success of the conversion. Two types of errors in conversion can occur, compile time errors and run time errors. Those errors detected at compile time are syntax oriented and will have to be addressed first. Once the converted application successfully compiles, run time errors can then be addressed. After the automated system problems are fixed, the engineers can move on to converting the next application in line.

5.4 System support Capability (effectiveness requirements)

In order to maintain the systems support capability to allow for the effectiveness requirement goals to be met the

following support mechanisms must be in place.

1. At least two engineers need to be available to perform conversion work each work day.
2. Spares have to be readily available within logistics for hardware repairs.
3. The ANSI C compiler will need to be delivered and ready for use by the time the automated system is operational.
4. After each application is converted it will be reviewed (tested) for the level of success of the conversion. If the Mct to fix code failures of the automated system will take much longer than 6 minutes, the software engineer must just simply fix the failure within the "custom made" application that was converted and abandon attempts to repair code within the automated system. This only applies to Mct during the operational phase of the system, not fix times during the systems test phase.

Without these support requirements there will not be efficient system support capability. It is important to make sure that at least one of the engineers involved with the use if the system have more than just a development perspective into Sybase but Sybase maintenance knowledge as well.

5.5 Maintenance Environment

The maintenance environment will be primarily internal to

the MMDC. The system will be produced internal to the MMDC. All maintenance pertinent to the system will be performed by software engineering, HMO, Sun Microsystems and Sybase.

6.0 Preliminary Design

6.1 Functional Analysis

Figure 3 on the following page starts a series of functional flow diagrams that facilitate the design, development, and system definition process in a logical manner. This is known as a functional analysis. The functional analysis is based on the definition of system operational requirements and system maintenance concept.[1]

The life cycle of the project will encompass the following four phases shown Figure 3. below. The second level of the design is broken down into both hardware and software.

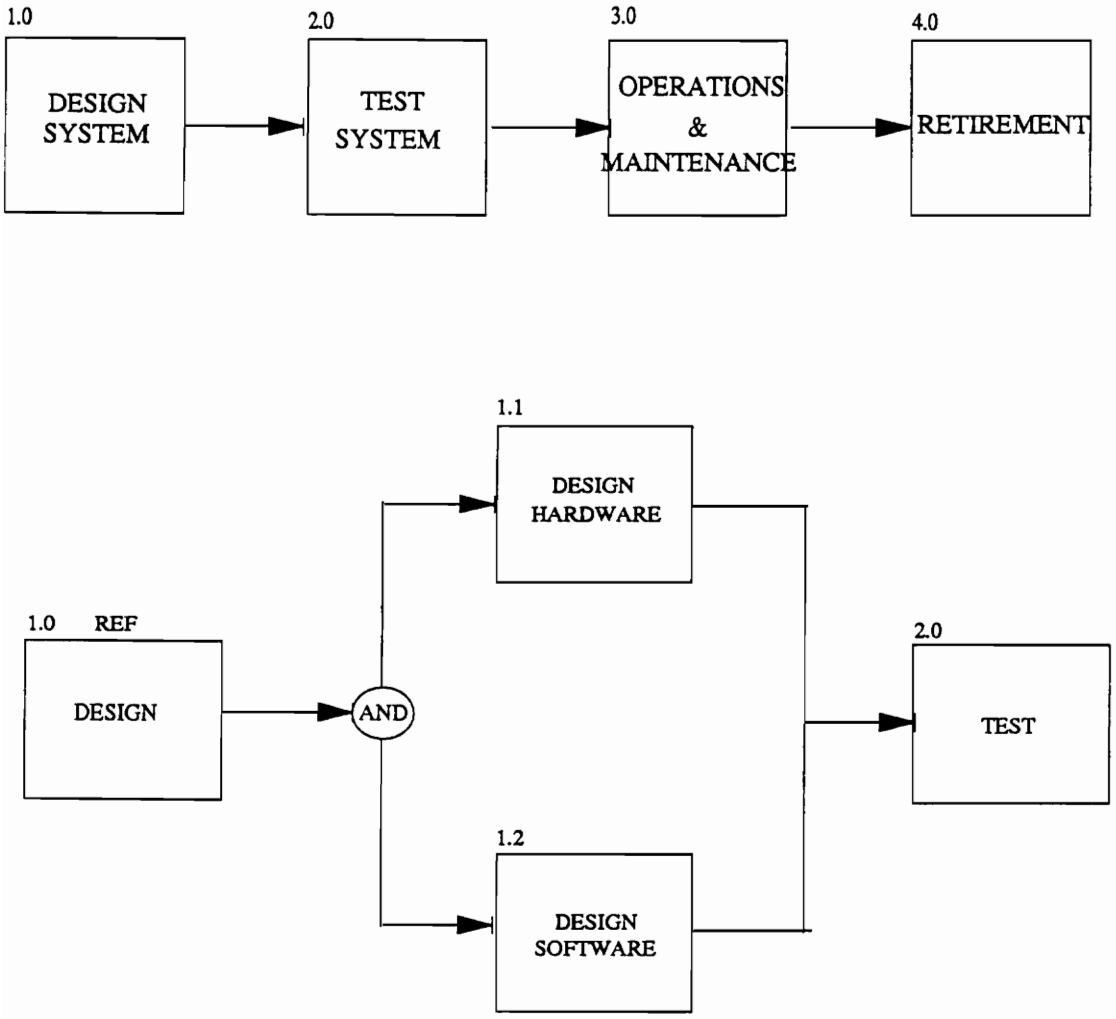


Figure 3. Level 1 & 2 Design Functional Analysis

Third level software design is expanded in Figure 4. A portion of the fourth level functional flow is also shown below.

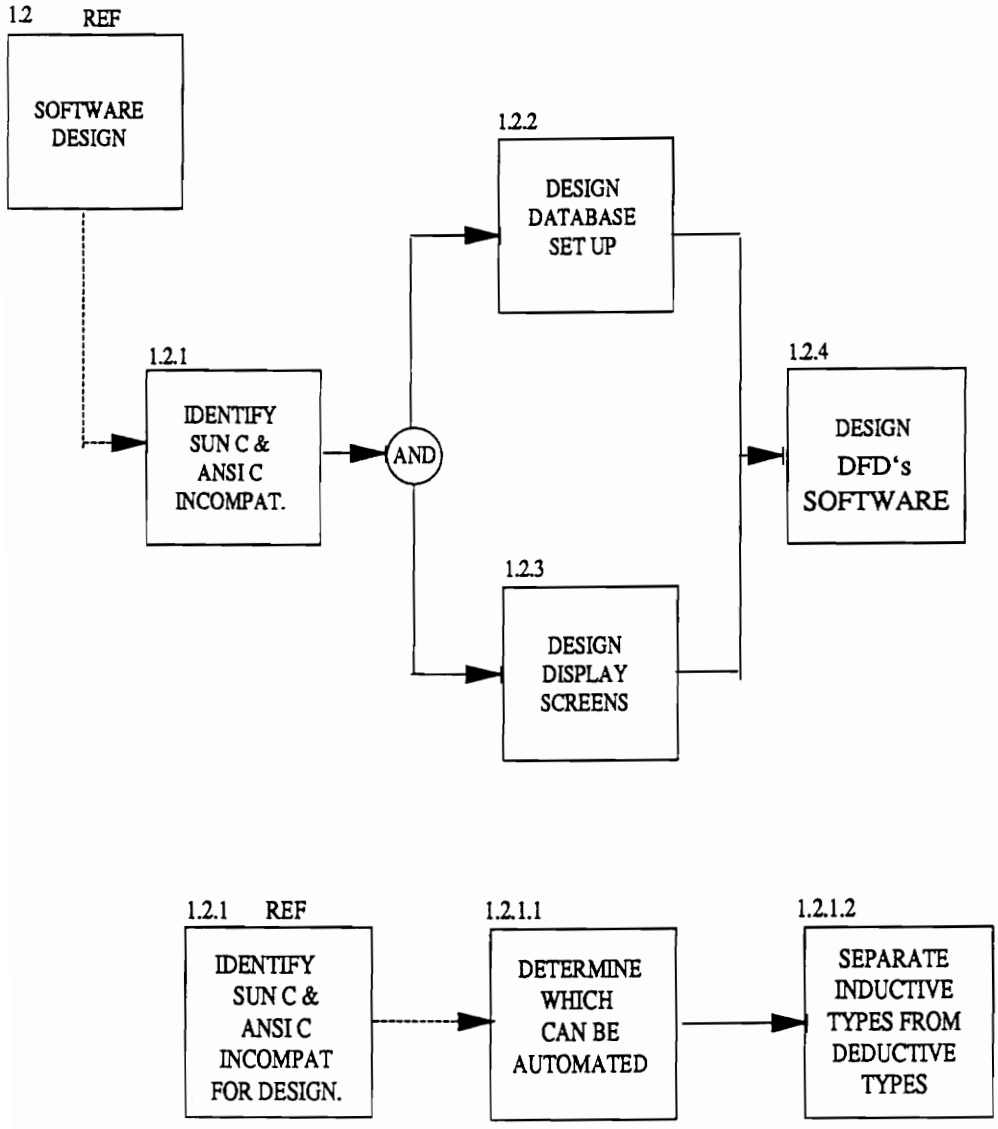


Figure 4. Level 3 & 4 Software Design

Part of the software design will be to set up a database with tables based on the requirements needed and design any display screens. The flow below shows the order in which tasks will be accomplished.

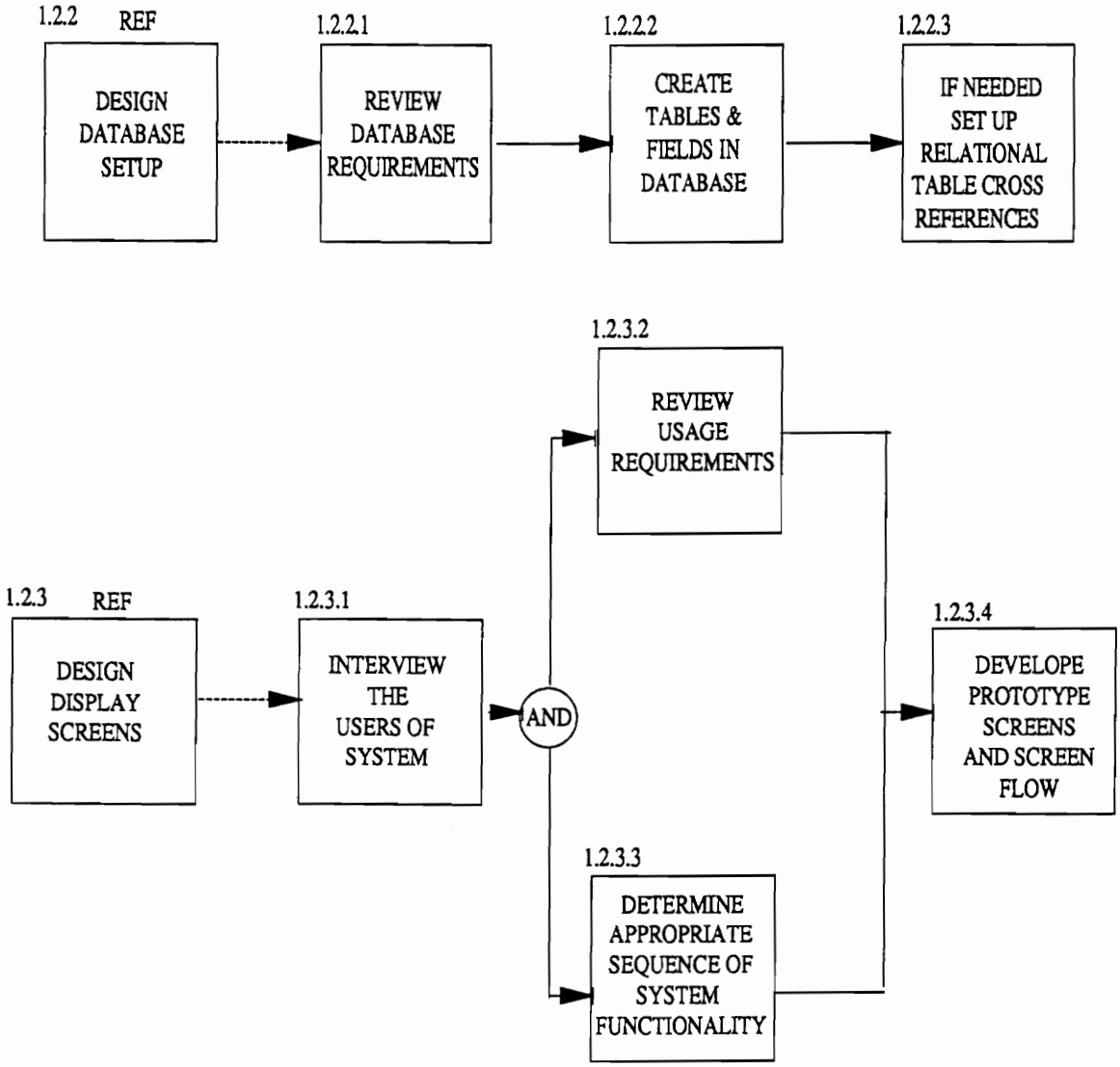


Figure 5. Level 3 & 4 Software Design.

Fourth level software analysis is shown in Figure 6 below. Both inductive and deductive incompatibilities need to be evaluated. The functional flow of the tasks are shown.

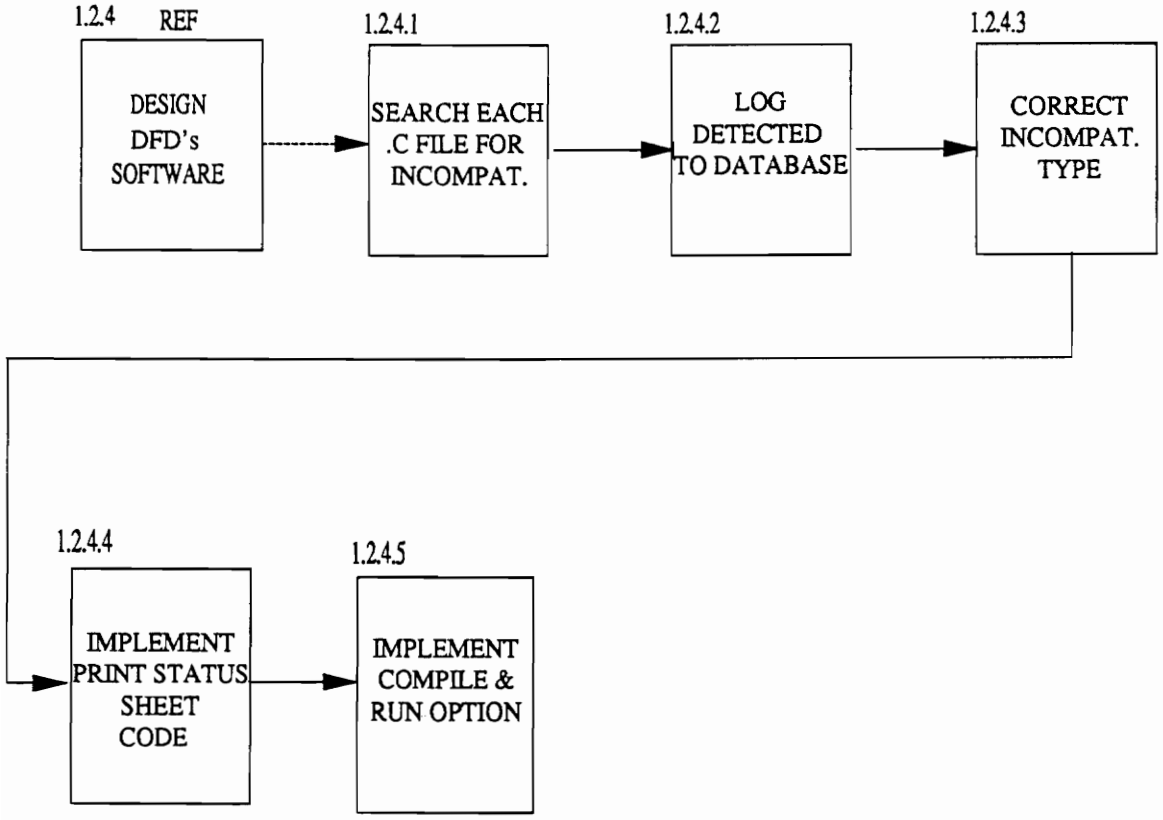


Figure 6. Level 3 & 4 Software.

Figure 7, below, further breaks down the hardware design into a third level analysis.

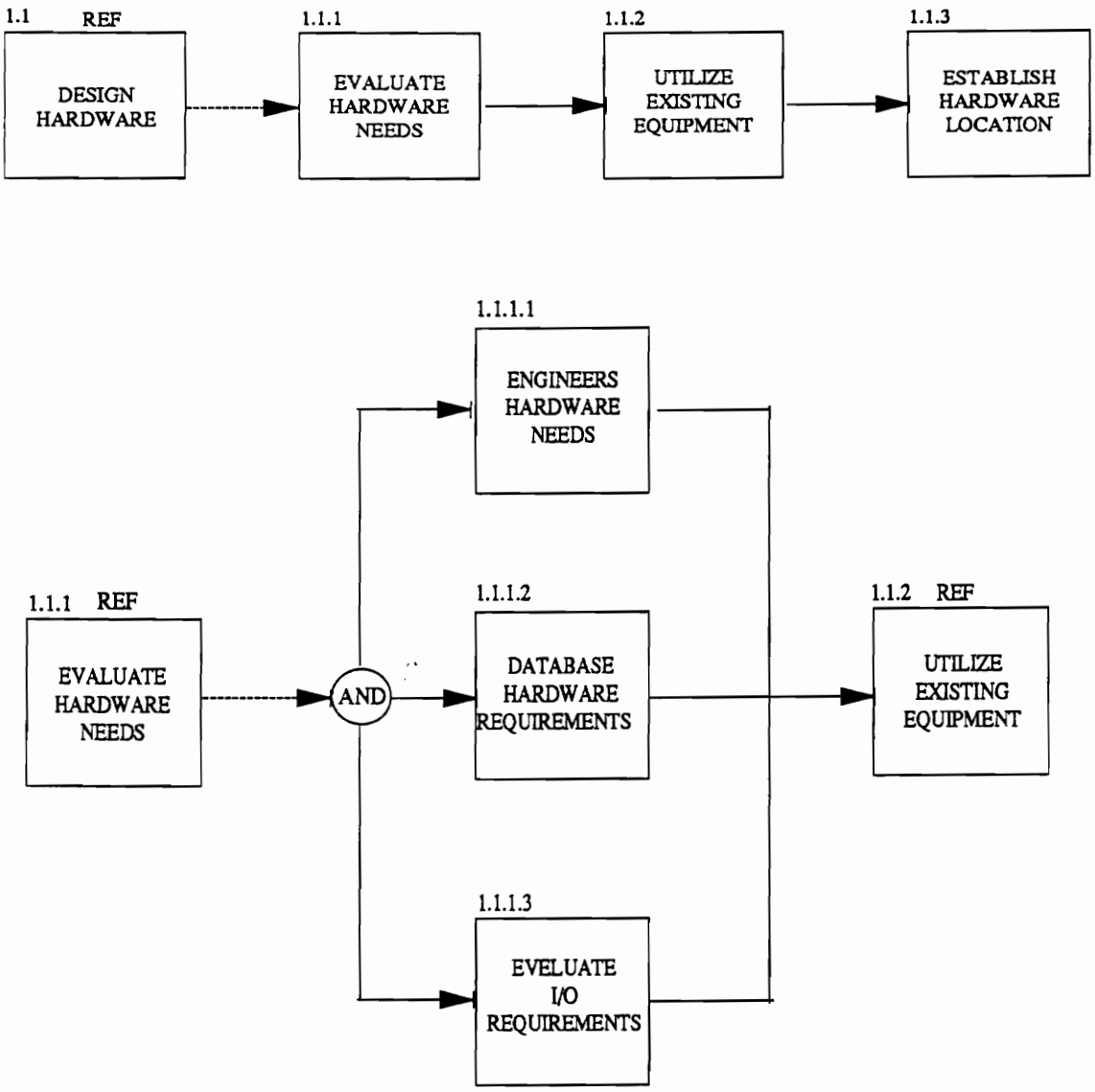


Figure 7. Level 3 & 4 Hardware Design

Figure 8 shows the fourth level breakdown flow of hardware design. Focus is on utilizing existing hardware within the MMDC and where within the facility the hardware will reside.

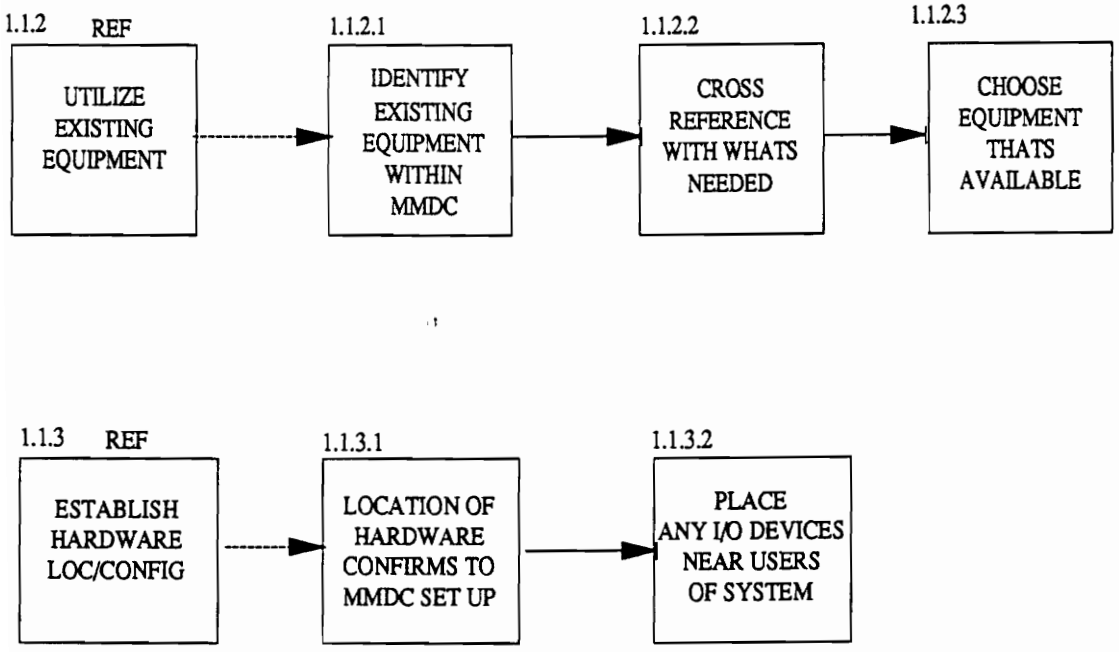


Figure 8. Level 4 Hardware Design.

After design is complete the system will under go a test phase. The areas tested will ensure that the automated system has met its performance requirements.

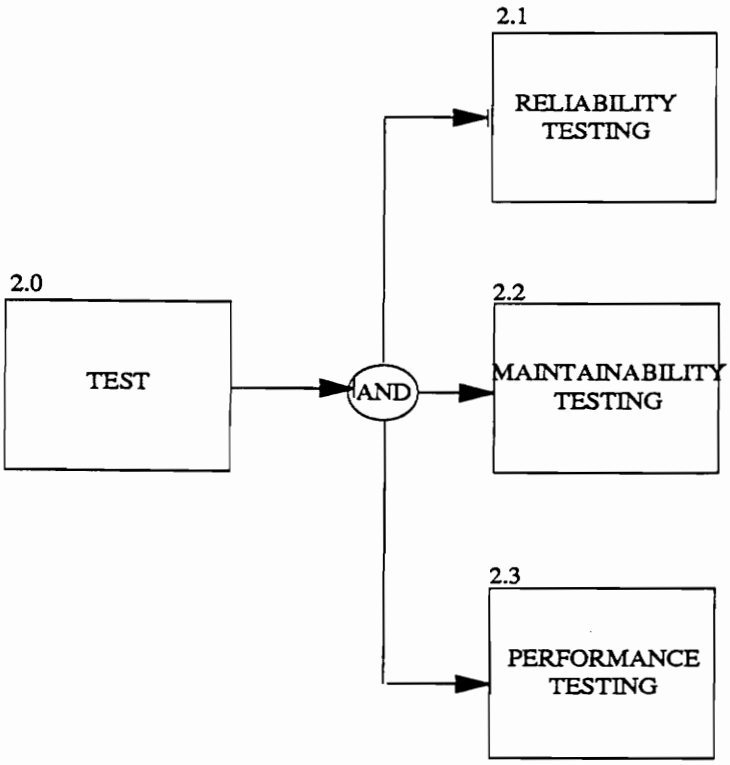


Figure 9. Level 1 & 2 Test.

Figure 10 shows the first level tasks needed for operations. Starting the automated system is actually expanded to the third level.

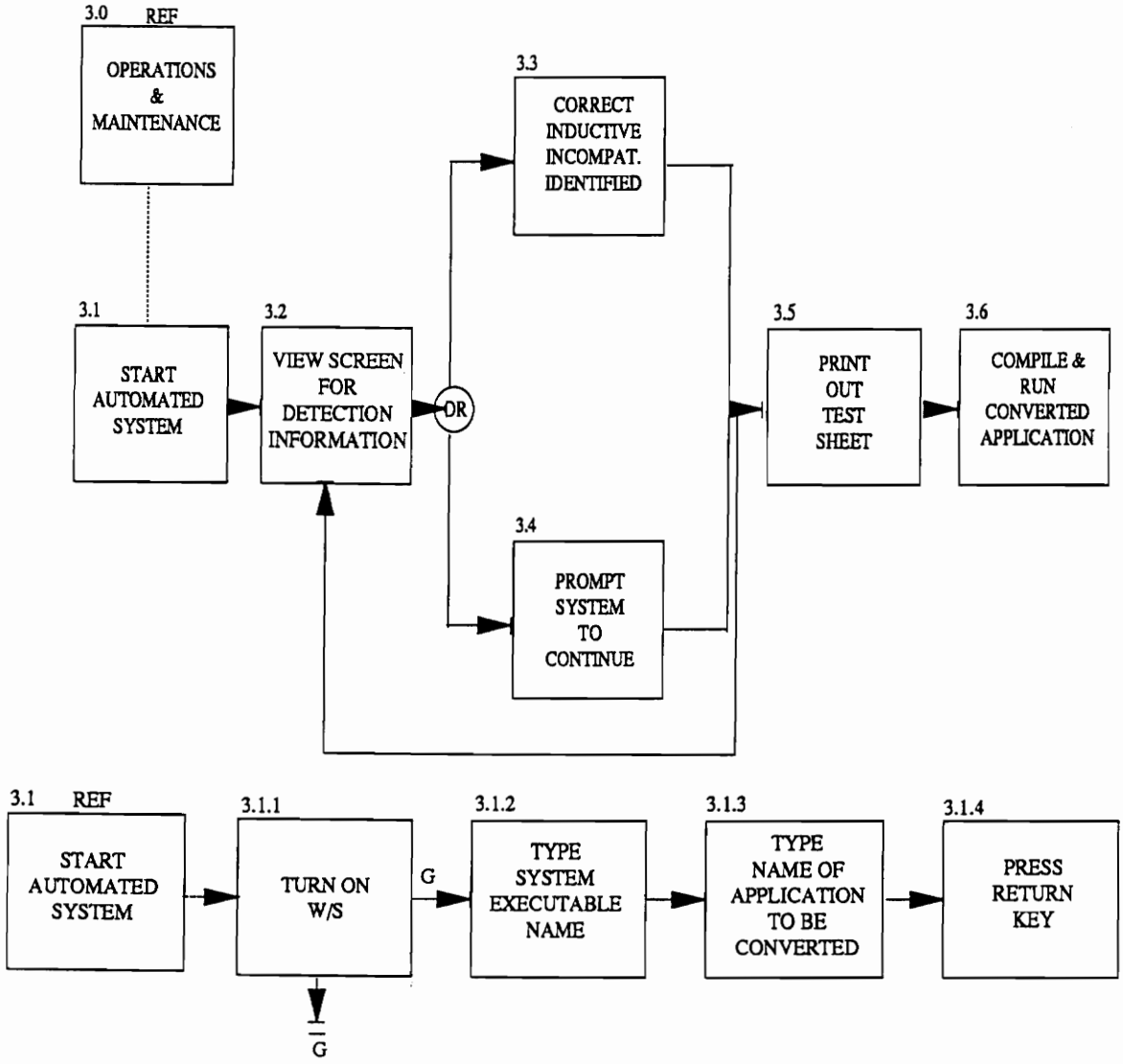


Figure 10. Level 2 & 3 Operations & Maintenance.

First level tasks needed for maintenance of the workstation and third level operational tasks are shown in Figure 11 below.

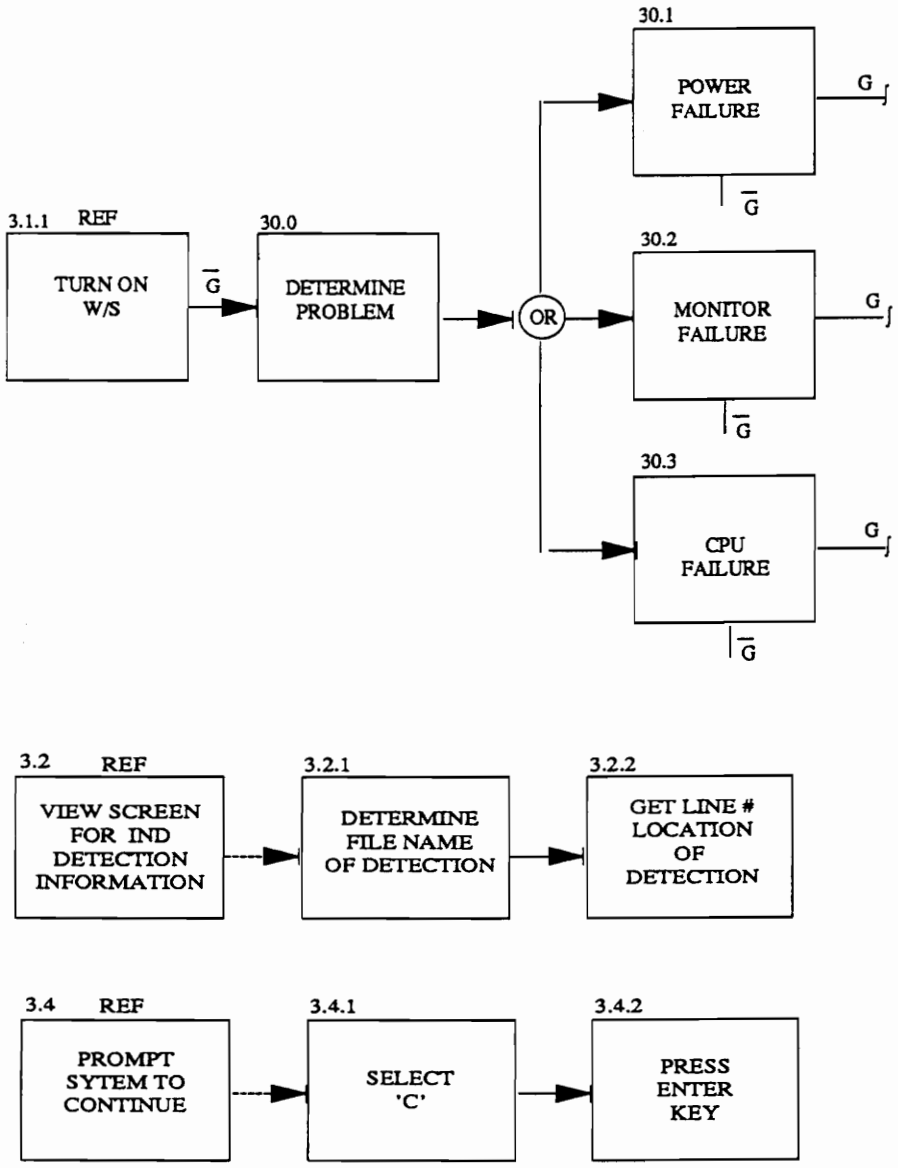


Figure 11. W/S Failure Maintenance & Level 3 Operational

The flow of the corrective maintenance tasks of the workstation hardware is shown in Figure 12 below.

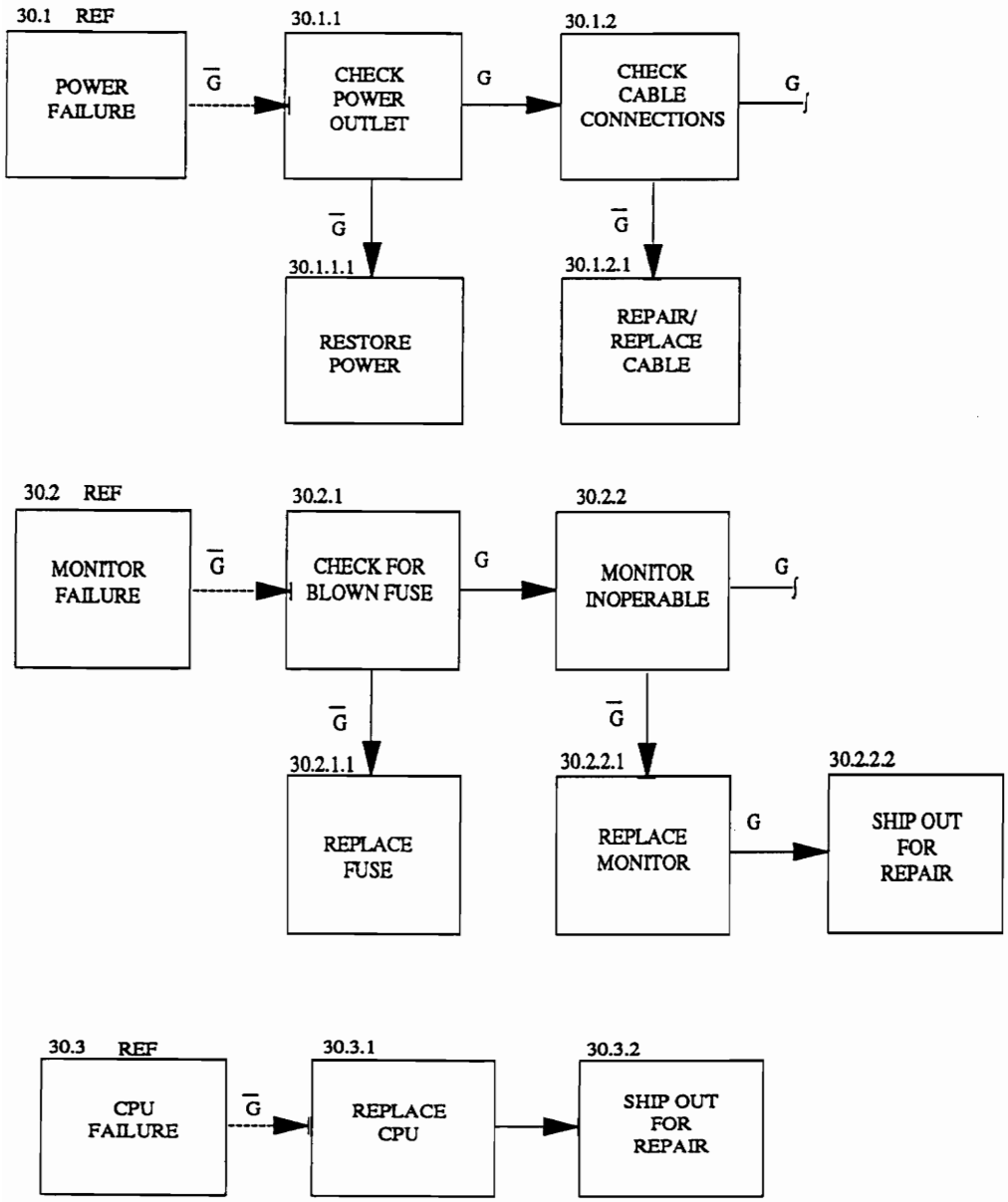


Figure 12. Monitor, Power & CPU Failure Maintenance.

The third level operational flow is shown in Figure 13 below.

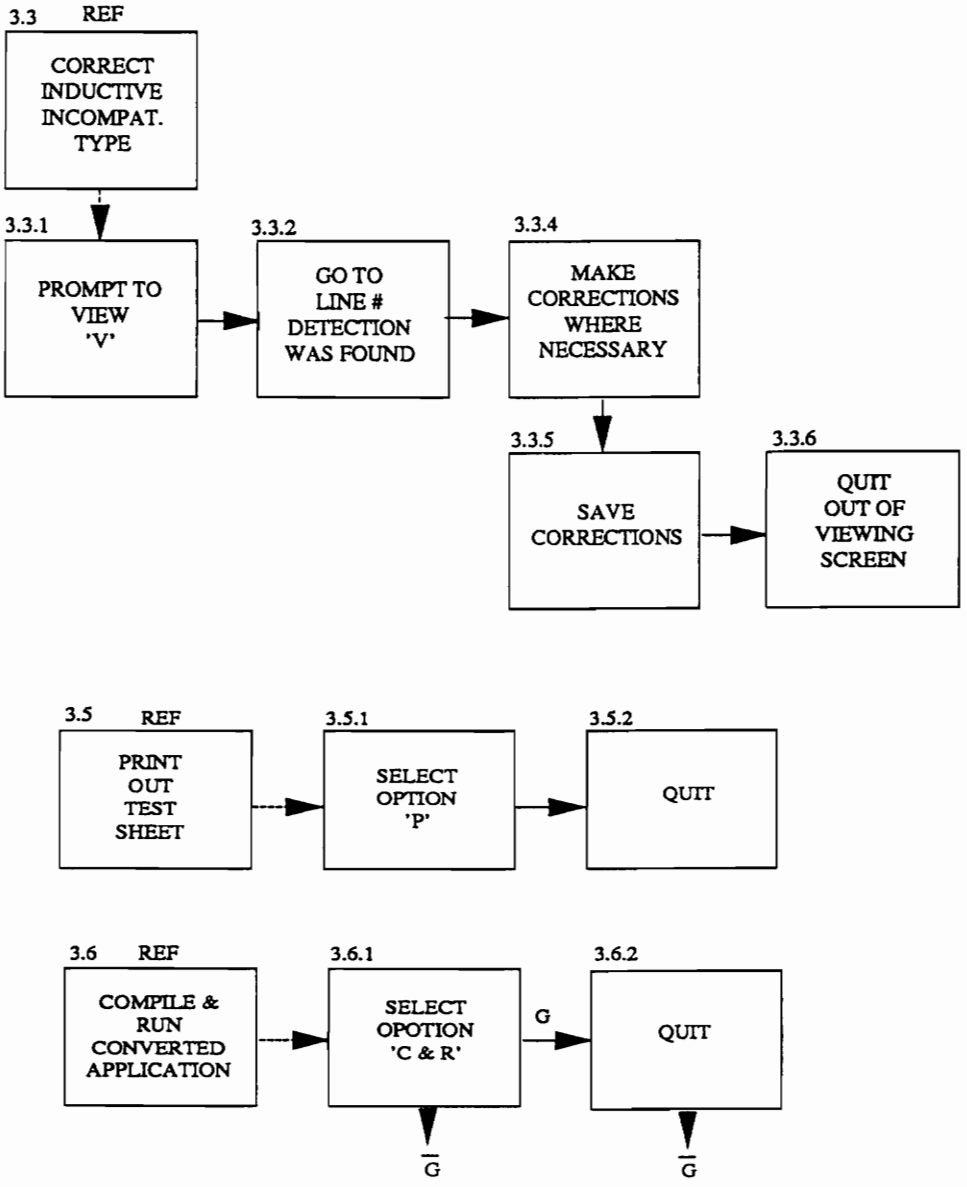


Figure 13. Level 3 Operational.

Corrective maintenance tasks are broken down and shown in Figure 14 below.

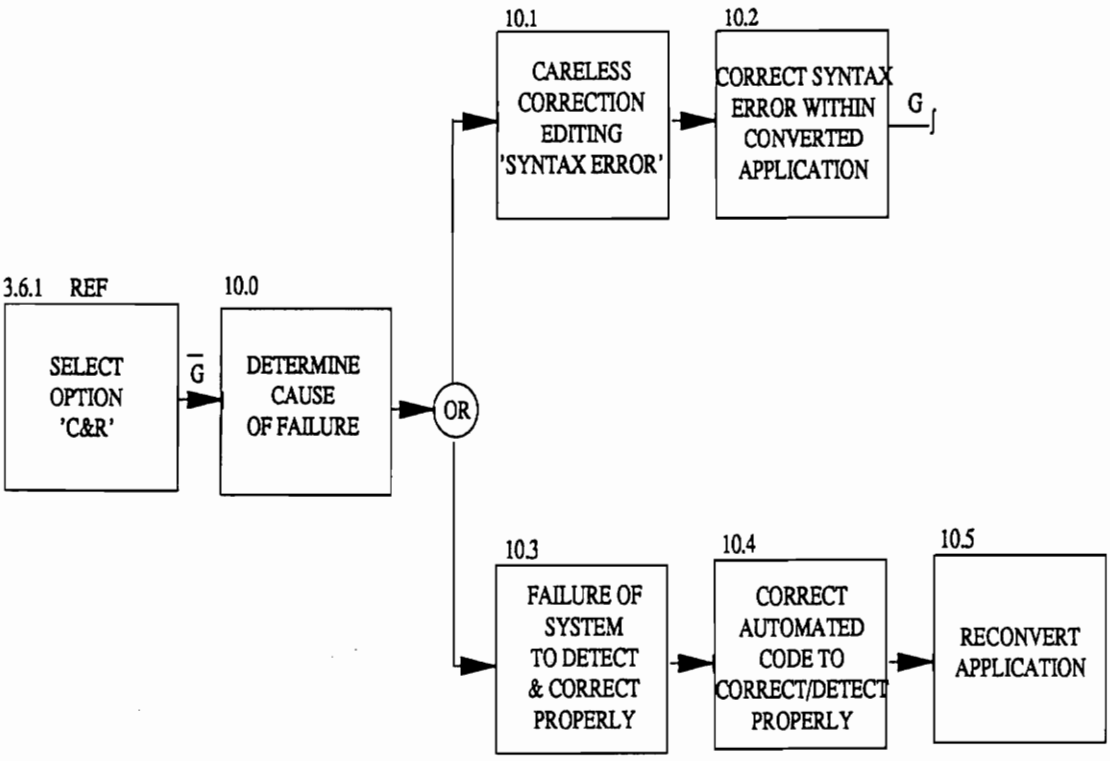


Figure 14. Compile & Run and Power Failure Maintenance.

The flow of corrective maintenance tasks for a hard disk failure are shown in Figure 15 below.

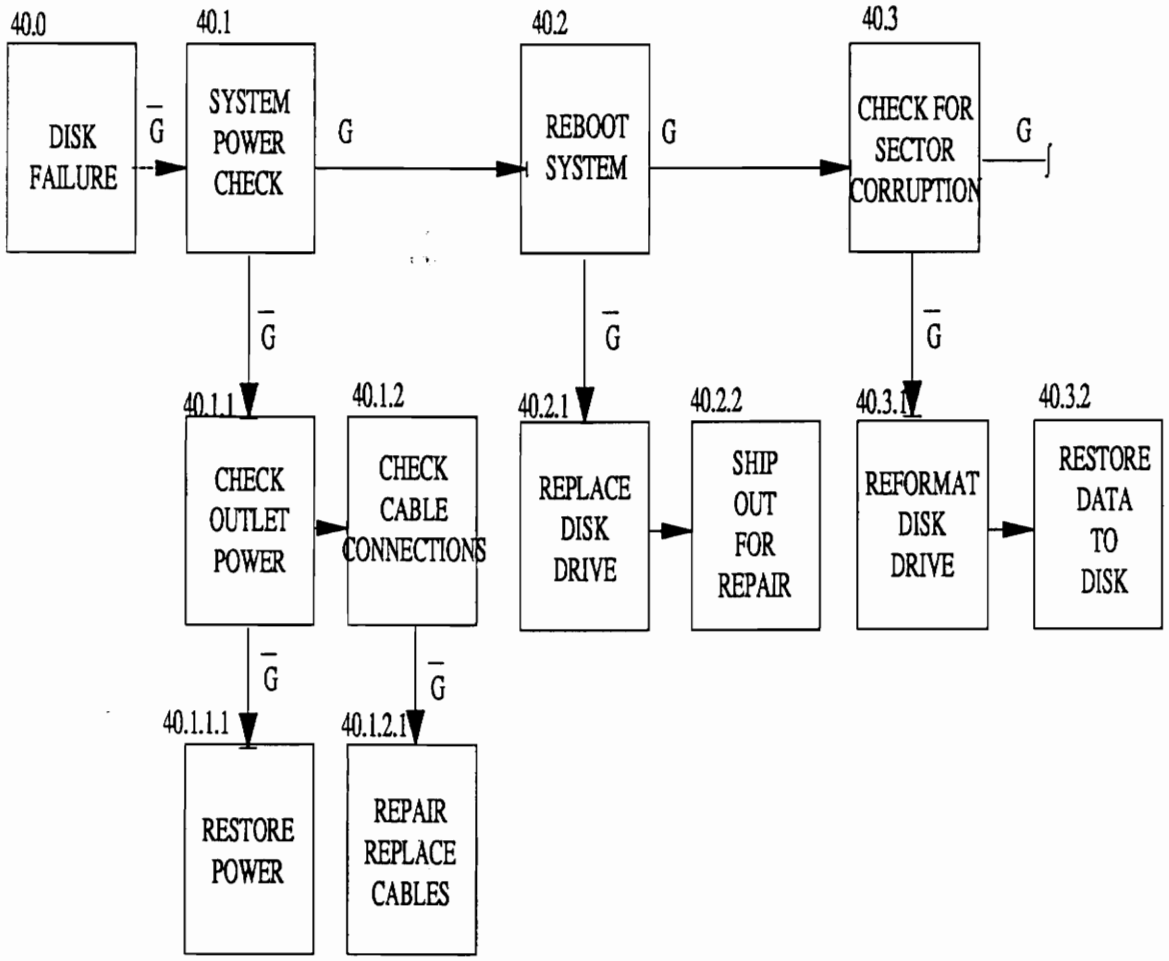


Figure 15. Disk Failure Maintenance.

Task flow for Corrective maintenance for the database is shown in Figure 16 below.

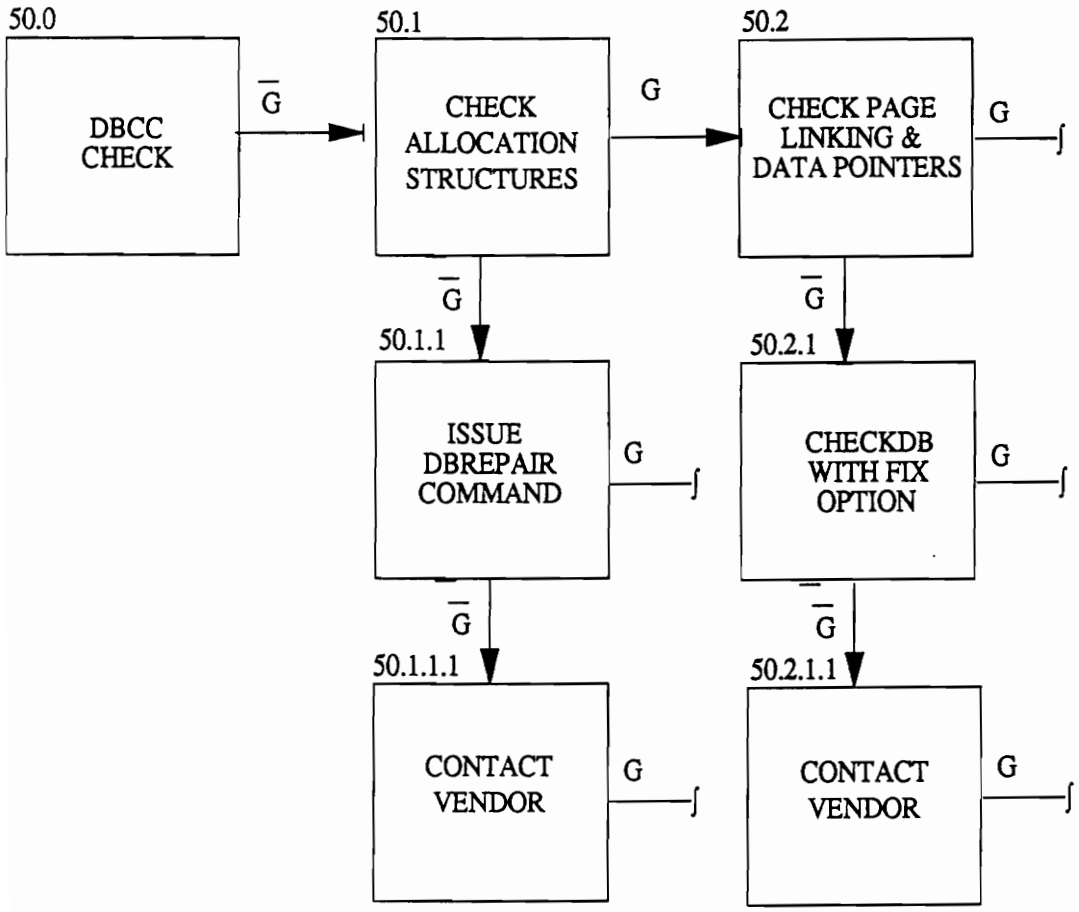


Figure 16. Database Maintenance.

6.2 Allocation of Requirements

6.2.1 Explanation of Technical Performance Measures (TPM's)

The following paragraphs give a description of each TPM of the system. An explanation of how accurate performance is to be measured is given. The MTBF of 2.2 hrs for the converter software and the system availability of 95% are requirements.

6.2.1.1 Measurement of Accurate Performance Requirement

Let the variable TCON represent the total number of incompatibilities detected upon completion of the conversion of an application. Let the variable TFAIL represent the total incompatibilities that failed to get converted or detected properly. Let TOTAL represent the sum of both TCON and TFAIL. Finally, Let the variable AP represent overall accuracy that the system performed in converting the application. The measurement of accurate performance can then be determined by

AP = (TOTAL - TFAIL)/TOTAL. (Eq. 6)

This is the equation that the test engineers will utilize when determining the actual accuracy of an application conversion.

6.1.1.2 Mean Time Between Failure (MTBF) Requirement

From section 2.2 it was determined that one incompatibility exists per 20 lines of code. Given that the measurement of accurate performance is 98%, the automated system, operated by two software engineers, will detect 331 incompatibilities per 6625 lines of code, (98% of 331 is 324).

This yields a software failure rate, of the converter software, of 7 failures per 8 hours, 3.5 failures for each individual each day. Thus in an 8 hour day

MTBF = 2.2 hrs. (Eq. 7)

6.1.1.3 Availability Requirement

There are three types of availabilities; inherent, achieved, and operational. The availability most applicable to the automated system is inherent. It excludes preventive or scheduled maintenance actions, logistics delay time, and administrative delay time.[1] The only maintenance actions that will be performed on the system are corrective maintenance actions that can only be addressed upon detection of a failure. At 3.5 failures per engineer a day (8 hours) with a MCT of 6 minutes. This yields an availability of 95.6%. The availability formula is

Ai = MTBF/(MTBF + MCT) (Eq. 8)

where MCT is mean corrective time. From Figure 17 on the following page,

Ai = 2.13/(2.13 + .108) = .95

6.2.2 How MTBF Relates To MTBM

Since the system is only concerned with inherent availability the systems MTBM will be equivalent to MTBF. By definition

MTBM = 1/(1/MTBMu + 1/MTBMs) (Eq. 9)

where

$$1/MTBMs = 0 \qquad \text{(Eq. 10)}$$

since there is no preventive or scheduled maintenance actions. MTBMu should approximate MTBF. Thus substituting MTBF into equation above

$$MTBM = 1/(1/MTBF) = MTBF \qquad \text{(Eq. 11)}$$

for the automated system.

6.2 Allocation Diagrams

The Mct of the system should be allocated to both the hardware and software unit levels down through their assembly levels. Table 1, 2, 3 on the following pages shows the Mct allocation breakdown for the both the system level and the unit levels. Figures 17, 18 and 19 display the allocation of requirements for the TPM's described above.

The System is comprised of both hardware and software components. The Software and hardware are subsystems of the overall system. The MMH/OH of the system is established by adding the MMH/OH of each subsystem together.

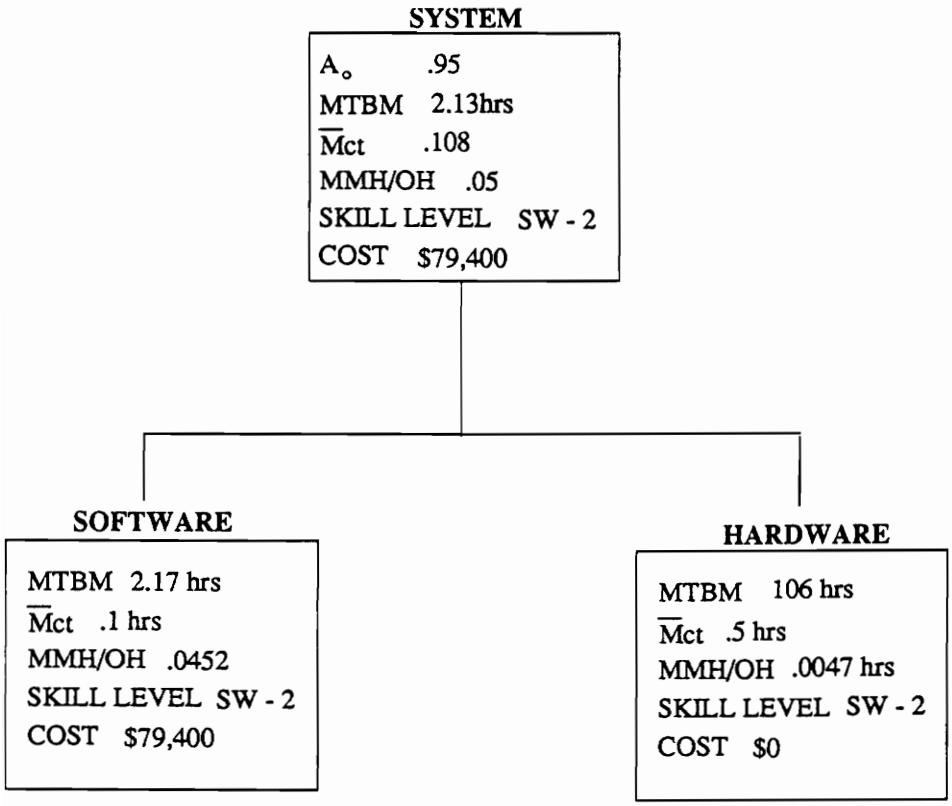


Figure 17. System Level Allocation.

TABLE I. MCT ALLOCATION FOR SYSTEM.

1	2	3	4	5	6	7
ITEM	QUANTITY OF ITEMS PER SYSTEM	FAILURE RATE (λ) X 106	CONTRIBUTION OF TOTAL FAILURES $Cf = (Q)(\lambda)$	PERCENT CONTRIBUTION $Cp = Cf \Sigma Cf X 100$	AVERAGE CORRECTIVE MAINT TIME $\bar{M}ct$(hr)	CONTRIBUTION OF TOTAL CORRECTIVE MAINT. TIME $Ct = (Cf)(\bar{M}ct)$
Software	1	48.84	48.84	98%	.1	4.884
Hardware	1	1	1	2%	.5	.5
Total			$\Sigma Cf = 49.84$	100%		$\Sigma Ct = 5.385$

$$\text{Mct for System} = \frac{\Sigma Ct}{\Sigma Cf} = \frac{5.384}{49.84} = .108$$

This table calculates the Mct of the system. The system hardware failures can be as much as 48.84 to 1 software failure over a span of 106 hours of the operational life cycle.

The allocation of requirements is a bottoms up approach. In the requirements section it stated that the Mct of the system hardware failures is thirty minutes. To achieve the appropriate Mct required for the system hardware, each hardware component was given a Mct of .5 hrs. The MMH/OH of the hardware subsystem is the sum of the MMH/OH of all three components. The MMH/OH of each hardware component is determined by MMH/OH divided by MTBM. Where the MTBM of this system will be equal to the MTBF.

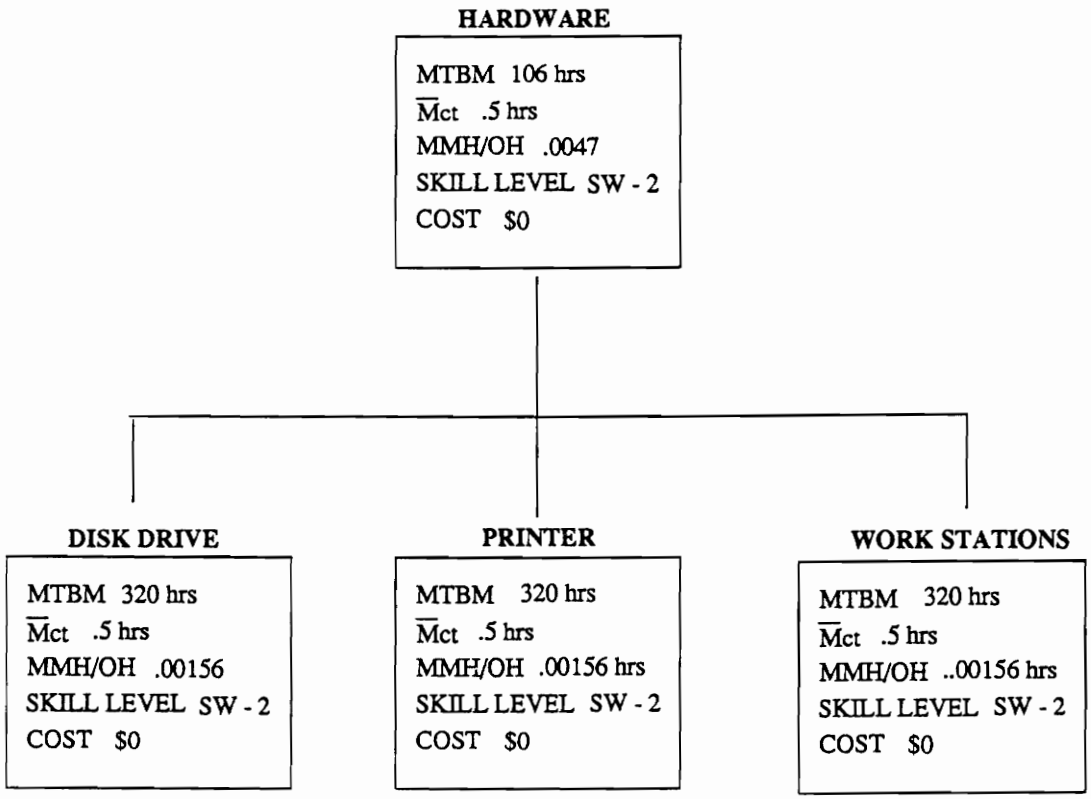


Figure 18 Unit Level (Hardware) Allocation.

TABLE 2. UNIT LEVEL MCT HARDWARE ALLOCATION.

1	2	3	4	5	6	7
ITEM	QUANTITY OF ITEMS PER SYSTEM	FAILURE RATE (λ) X 320	CONTRIBUTION OF TOTAL FAILURES $Cf = (Q)(\lambda)$	PERCENT CONTRIBUTION $Cp = Cf \Sigma Cf \times 100$	AVERAGE CORRECTIVE MAINT TIME $\overline{Mct}$(hr)	CONTRIBUTION OF TOTAL CORRECTIVE MAINT. TIME $Ct = (Cf)(\overline{Mct})$
Diskdrive	1	1	1	25%	.5	.5
Workstation	2	1	2	50%	.5	1
Printer	1	1	1	25%	.5	.5
Total			$\Sigma Cf = 4$	100%		$\Sigma Ct = 2$

$$\overline{Mct} \text{ for System} = \frac{\Sigma Ct}{\Sigma Cf} = \frac{2}{4} = .5$$

The Mct for the hardware subsystem is .5. Note that there are two workstations that could possibly fail at any given time. All hardware components have an estimated failure rate of only 1 for every 320 hours of operational life.

In the requirements section, it states that the MTBF of the developed converter software shall be 2.2 hours with a Mct of 6 minutes. Thus, these values have been allocated to the converter software. The chances of failures in the operating system software and the database are minimal, no more than 1 failure is expected every 360 hours. The MMH/OH of each software component is determined by MMH/OH divided by MTBM. Where, as mentioned earlier, the MTBM of this system will be equal to the MTBF.

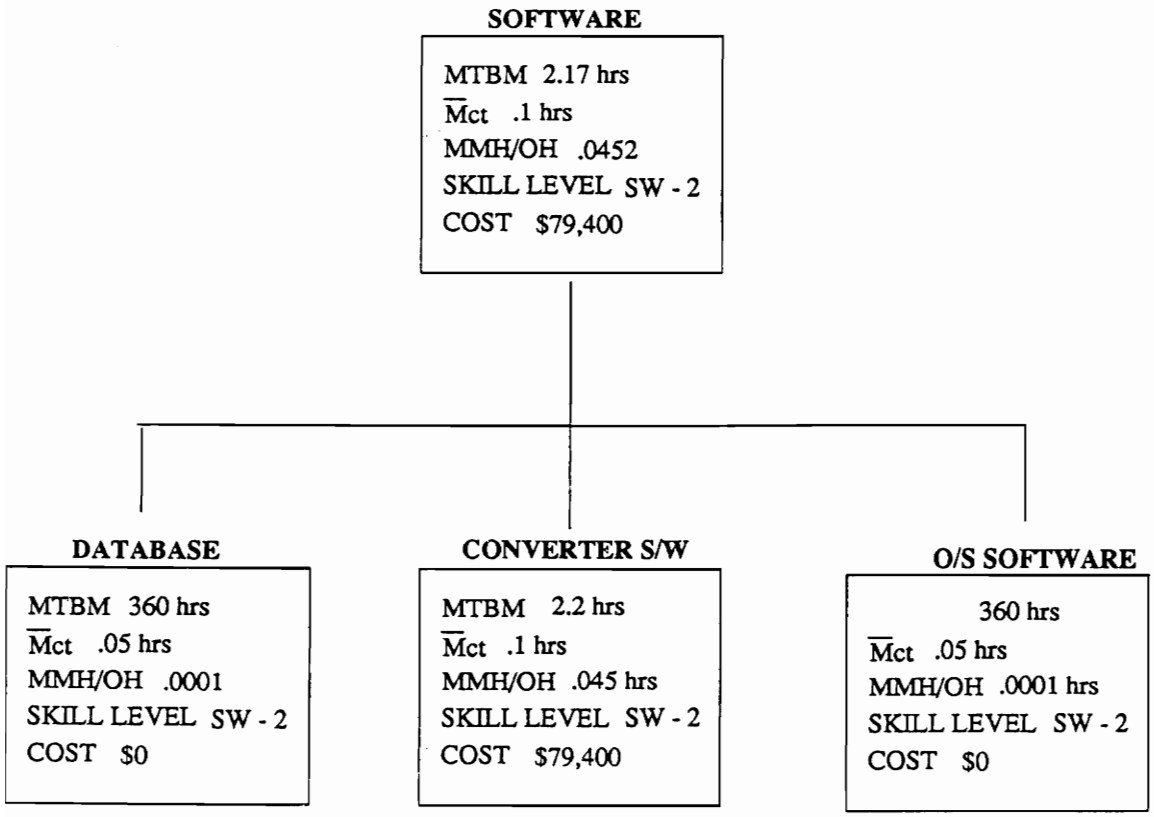


Figure 19. Unit Level (Software) Allocation.

TABLE 3. UNIT LEVEL MCT SOFTWARE ALLOCATION FOR SYSTEM.

1	2	3	4	5	6	7
ITEM	QUANTITY OF ITEMS PER SYSTEM	FAILURE RATE (λ) X 360	CONTRIBUTION OF TOTAL FAILURES $Cf = (Q)(\lambda)$	PERCENT CONTRIBUTION $Cp = Cf \Sigma Cf \times 100$	AVERAGE CORRECTIVE MAINT TIME $\bar{M}ct$(hr)	CONTRIBUTION OF TOTAL CORRECTIVE MAINT. TIME $Ct = (Cf)(Mct)$
Database	1	1	1	.6%	.05	.05
Converter S/W	1	163	163	98.8%	.1	16.3
O/S	1	1	1	.6%	.05	.05
Total			$\Sigma Cf = 165$	100%		$\Sigma Ct = 16.4$

$$\bar{M}ct \text{ for System} = \frac{\Sigma Ct}{\Sigma Cf} = \frac{16.4}{165} = .099\ (.1)$$

The Mct for the software subsystem is .5. Note that there are 168 converter software failures to only 1 O/S or Data-base failure over the course of a 360 hour time frame. This yields an overall Mct for the software subsystem of .1 hrs.

7.0 Detailed Design Phase

7.1 Design Hardware

7.1.1 Evaluate Hardware Needs

The hardware that is to be utilized in the development of the automated system will already exist within the MMDC. It is required that the development team have 2 sparc workstations, utilize a F/S disk for the database and a printer to print out pertinent test information.

7.1.2 Utilize Existing Equipment

The workstations that exist within the MMDC are sparc ELC's and Sparc 2's. Since the Sparc 2's have enhanced processing capabilities they will be the workstations of choice for development. The 4/75 file server that the software engineering group is hosted to has available disk space. Thus this will be the File server disk that the database of the system will utilize. The only printers that exist within the MMDC are QMS 810's. One of these printers will be used for any printing capabilities of the system.

7.1.3 Establish Hardware Location/configuration

Within the MMDC, as discussed in the need statement, each file server supports a maximum of eight W/S. All file servers are located on the third floor and all W/S are spread throughout all rooms within the facility. Since five of the eight workstations utilized within the software engineering group are Sparc 2 workstations, two of them will be designated

to the development of the automated system. Since the workstations already reside within the software engineering group, where development work will be done, their is no need to relocate them. An additional printer will need to be placed within software engineering when testing of the system begins. This printer will be pulled from logistics and returned once testing is complete. The workstations will need to be connected to the appropriate 4/75 file server. Figure 20. on the following page shows the hardware location/configuration needed for development of the automated system.

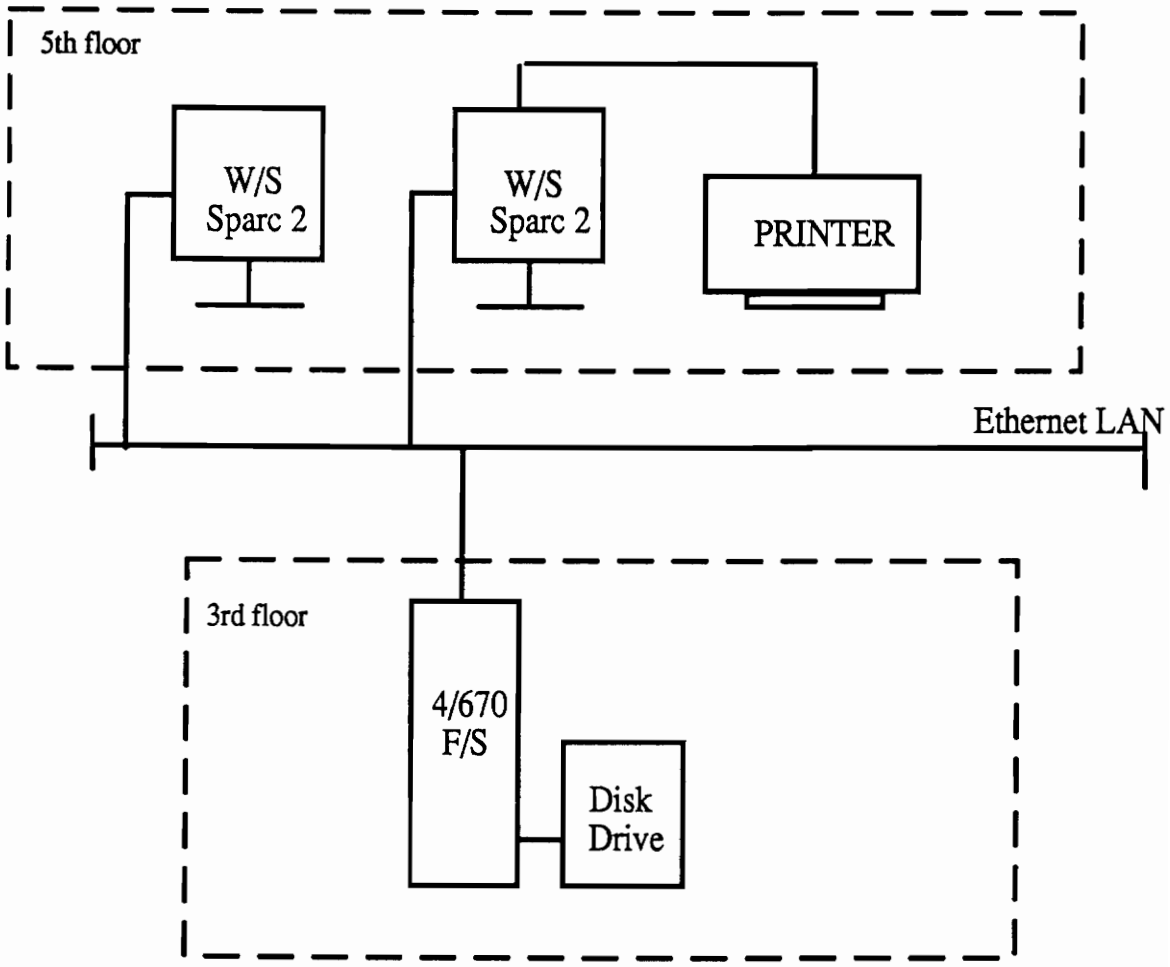


Figure 20. Hardware Design Configuration.

7.2 Software Design

7.2.1 Design Database Set Up

Currently, the database that the MMDC utilizes is SYBASE. Since the requirement is to utilize existing hardware and software where ever possible, the database for use by the automated system will be Sybase. There are three tables needed for the database. Database tables 1, 2 and 3 will be logged to at different times with different information. Table 1; every time an incompatibility is detected, Table 2; every time 6625 lines of code is traversed and checked for all 20 incompatibility types, and finally Table 3; the incompatibility totals will be logged upon completion of an application conversion. The table names along with their field names are described in the sections below. Next to each field a brief description of the fields purpose is stated and what datatype of information is allowed within the field.

7.2.1.1 Table 1 Design

The purpose of this table is to allow the requirement, which states that 6625 lines of code must be traversed per day, to be tested. The date/time and a cumulative record count, in increments of 6625, will be logged to this table by the automated system every time 6625 lines of code are traversed while converting an application. Once the conversion of the application is complete a "Performance Status Sheet" will be printed out that contain these values.

From the sheet it can be determined if 6625 lines of code are being traversed in an 8 hour day. Listed below are the fields that will exist in this table.

Application_Name:

This will identify the "custom made" application converted. This field will be of *char* type with a maximum length of 30 characters.

Datetime:

This will be a log of the exact time that the automated system traverses through 6625 lines of code. This field will have datatype *datetime*.

Cumulative_Line_count:

This field will contain the amount of lines being traversed, for the application being converted, in increments of 6625. The field will have datatype *int*.

Time_elapsed:

This will designate the amount of time, in hours, that it took the automated system to traverse through 6625 lines of code. The field will be of type *int*.

7.2.1.2 Table 2 Design

This table will address the requirement that each incompatibility type detected is to be logged to the database. Records are logged to this table every time an incompatibility is detected by the automated system. This table will document all the incompatibilities that were detected within an

application. Simple queries of the database will reveal what file and where in the file incompatibility types were found.

Application_name: Described in table 1 above.

File_name:

This field contains the name of the file where the incompatibility was found. This field will be a 30 character length field of type *char*.

Line_number:

This field will contain the line number within the file, identified above, of where the detection occurred. This field will be of type *int*.

Incompat_id:

There are 20 different types of incompatibilities identified from Sun Microsystems that could exist within Sun C code. They have been assigned id values that range from 1 to 20. The Incompat_id field will be of type *int* and contain any value from 1 to 20.

Incompat_type:

When an incompatibility is detected it will be stated as either inductive or deductive (ind or ded). This establishes whether correction of the incompatibility detected was automated or the engineer had to make the actual correction. This field will be of *char* type field of 3 character lengths.

7.2.1.3 Table 3 Design

The purpose of this table is to satisfy the requirement to log the total number of incompatibilities found. The total will be used to test how accurate the system was in the conversion of an application. Records are logged to this table once the system has completed a conversion of an application.

Application_name:

Described in table 1 above.

Total_detected:

This field will hold the numeric value of the total # of incompatibilities that were detected in the conversion process. It will be of datatype *int*.

Ind_type_cnt:

Tells how many inductive type incompatibilities were detected. This field will be of type *int*.

Ded_type_cnt:

Tells how many deductive type incompatibilities were detected. This field will be of type *int*.

7.2.2 Design Display Screens:

In order to determine the appropriate screens that need to be displayed the designer needs to know what the users need. The users of this system are software engineers. Ease of use, at a software engineering level, is what is important when determining the screen interaction of the automated system since they will be the users. The engineers expressed

a desire that the system shall have as little screen software development as possible. Upon review of system requirements, existing software capabilities are to be utilized where ever possible. Since text editors are what is utilized everyday for the editing of software code, this tool will be incorporated into the automated system. From the above considerations the following sequence of screen displays and flows will be designed.

1. The following screen, shown in Figure 21. below, should appear once the automated process is started.

Enter the Makefile path of the application to be converted after the prompt below:

^:

Type Q and press return to Quit

Figure 21. Initial Screen.

2. The system will begin to convert application code until an inductive type of incompatibility is detected. Once it is detected the following information is displayed to the screen.

An inductive type of incompatibility has been detected in file XXXX on located on line XXXX within file.

View/edit file (V) Continue (C) Quit (Q)

^:_

Figure 22. Inductive Detection Screen

3. Upon selection of "V" the following sequence of events occur.
   - File comes up for editing via a text editor.
   - File can be edited and saved utilizing text editor commands.
   - Quit out of text editor when edits are complete.
4. Upon selection of "C" the automated process will continue on the conversion process. Two things can happen; Another inductive type incompatibility is detected and the screen described in step 2 will be displayed and steps 2 and 3 get repeated or the conversion will have completed and the screen below will be displayed.

Conversion Complete!

Select one of the following options:

Print out detection count (P)

Compile & Run Application (C&R)

Quit (Q)

^:_

Figure 23. Conversion Complete Screen.

5. Upon selection of (P) the following information will be printed to the screen and a "performance status sheet" will then be printed from the printer.

Application Name: XXXX

Number of incompatibilities detected: XXXX

Number of inductive type: XXXX

Number of deductive type: XXXX

Figure 24. Incompatibility Totals

6. Upon selection of (C) the system will compile the application just converted and compiler errors will be displayed to the screen "if any". The system will kick off the converted and compiled application executable to determine if any run time errors exist.

7. Upon selection of (Q) the engineer is able to quit out of the application.

7.2.3 Identify Sun C Incompatibilities with Sun ANSI C

The following describes Sun C features that are incompatible with Sun ANSI C. The list of incompatibilities was provided to the MMDC from Sun Microsystems.

7.2.3.1 Deductive Types

The following incompatibilities are labeled as deductive. This means that the automated system will both detect and correct the incompatibilities listed below. Following the explanation of each incompatibility will be a brief description as to how the automated process will detect and correct the incompatibility.

7.2.3.1.1 Keywords:

Three new keywords, known as type modifiers, have been added to ANSI C. They are known as **const**, **volatile**, and **signed**. Sun C considered them ordinary identifiers, not keywords. Data types that cast an identifier as type *char*, *short*, *int*, or *long* can be preceded by these modifiers

described below.

const:

The identifier can only be initialized once and can never be changed.

volatile:

The identifier can be updated to contain different values when needed throughout the program.

signed:

The identifier can have a signed representation.[2]

The new change is significant since old Sun C programs that have used the keywords above as ordinary variables (identifiers) will have to be changed.

Detection:

Global or local variables found within Sun C code named as either of the keywords stated above will be flagged as an incompatibility detection. The id for this detection is 1.

Correction:

Append the variable and all its occurrences with xxx. This will allow the variable to become a regular identifier and not a keyword.

7.2.3.1.2 Identifier Problem

Sun C allows dollar signs($) in identifiers, while ANSI C does not.

Detection:

Any identifier that contains a ($) will be flagged as an

incompatibility detection with id 2.

Correction:

All dollar signs will have to be removed from all identifiers and future references to the identifier. The dollar sign will need to be replaced with a unique value to not run the risk of becoming the name of another identifier once the dollar sign is removed.

7.2.3.1.3 Long Float

Sun C accepts **long float** declarations and treats them as double(s). ANSI C does not accept these declarations.

Detection:

Any declarations that are declared as **long float** will be flagged by the automated system as an incompatibility detection with id 3.

Correction:

All **long float** declarations will need to be converted to type *double*.

7.2.3.1.4 Assignment Operators

Sun C treats the following operator pairs as two tokens (tokens being basic units of compilation or smallest units of recognizable meaning) and will allow white space between them:

- *= (multiplication)
- += (addition)
- /= (division)
- %= (remainder)

- -= (subtraction)
- <<= (left shift)
- >>= (right shift)
- &= (bitwize "and")
- ^= (bitwize "exclusive or")
- |= (bitwize "inclusive or")

ANSI C treats them as single tokens and does not allow white space between them.[3]

Detection:

Assignment operator pairs that have white space between them (ex. * =) within existing Sun C code will be flagged with id 4.

Correction:

Simply remove the white space.

7.2.3.1.5 Multi-byte character constants

Multi-byte representation of characters is needed for Asian language character sets which contain thousands of characters. All character constants have type *int* and the value of a character constant is the numeric value of the characters ASCII representation. Sun C and ANSI C represent multi-byte constants differently. For example:

```
int mc = 'abcd';
```

yields abcd in Sun C and dcba in ANSI C.

Detection:

The key in detection a detection of this type is to

recognize identifiers declared as type int equated to a string of characters inside of single quotes. Sun C code that contains such declarations will be flagged and given id value of 5.

Correction:

The order of the character sequence defining the value of the variable will need to be reversed. This is necessary to preserve the intended value the programmer wanted for that variable.

7.2.3.1.6 Implied Integer Declarations

The Sun C compiler supports declarations without an explicit type specifier. In ANSI C this is not supported. Implicitly declared identifiers are not preceded by a datatype, for example:

```
num;        /* Implicitly declared as type int */
```

and

```
int num;  /* Explicitly declared as type int */
```

The Sun C compiler assumes that num is an integer even though it has not been explicitly declared as one. ANSI C compiler would generate a syntax error upon compilation of code.

Detection:

Variables followed by a ; will be considered as an implicit declaration. Detection of such declarations will be flagged with id 6.

Correction:

The identifier (num) will be explicitly declared as type *int* (integer).

7.2.3.1.7 Empty Declarations

Sun C compiler allows empty declarations, as shown in the example below:

```
int;
```

Since there is no identifier the declaration is considered empty and just ignored. ANSI C will not compile this type of declaration.

Detection:

Datatypes followed by a ; will be flagged with id 7.

Correction:

Declarations such as these will be removed from the code since they contribute nothing to the program.

7.2.3.1.8 Type Specifiers On Type Definitions

The Sun C compiler allows type specifiers such as *unsigned*, *short*, and *long* on *typedef* declarations. A *typedef* is a handy way to rename a lengthy declaration type so that a shorter version of the declaration can be used instead. For example:

```
typedef int    *ptr;
```

ptr is representative of type integer pointer and future declarations of identifiers that are to be of integer pointer type can use the *typedef* declaration *ptr* instead.

```
ptr link; /* link is declared as an integer pointer */
```

Sun C would let you further modify the typedef declaration as:

```
long ptr link;
```

Sun ANSI C will not let you do this.

Detection:

Any *typedef* declarations that have been further modified by type specifiers will be flagged with id 8.

Correction:

The *typedef* variable will be replaced with its defined declaration type.

7.2.3.1.9 Token Pasting Or Concatenation

A Token is defined as the basic unit of compilation or smallest unit of recognizable meaning such as operators, keywords, constants, and identifiers.[4] In Sun C, token concatenation in preprocessor operations was to use a comment in-between tokens

```
#define CONC(x,y) x/**/y
```

Where /**/ is the open and closing comment syntax and is Sun C's preprocessor operator for concatenation. The tokens being x and y.

Thus,

```
CONC(A,B)
```

in a program would yield the token AB. ANSI C does not support this syntax. Instead ANSI defines ## as the preprocessor operator for concatenation.

```
#define CONC(x,y)   x##y
```

ANSI C will treat the comments used in Sun C as white space yielding A B.

Detection:

Any two token in a preprocessor statement separated by /*any comment*/ will be flagged with the id 9.

Correction:

In order to correct this incompatibility with Sun ANSI C, all comments that reside between two tokens in a #define statement will have to be replaced with the appropriate preprocessor operator ##.

7.2.3.1.10 Stuff After #Else and #endif

The Sun C compiler would ignore anything on a line after a #else or #endif.[5] Many people used this to indicate which #if the construct was tied. For example:

```
#ifndef VAR
     compute_1();
#else
     compute_2();
#endif VAR
```

The VAR after #endif is the equivalent of a comment to show which #if statement it belong to. Sun ANSI C eliminated this feature.

Detection:

Trailing information detected after the #else or #endif will be flagged with id 10.

Correction:

In order to solve this code difference the two trailing tokens will be treated as comments by placing /* before the token and */ after the token.

7.2.3.1.11 Treatment Of Tags In Incomplete Declarations

Sun C ignores incomplete structure type declarations shown as,

```
struct x;      /* Incomplete */
```

In ANSI C this incomplete declaration would hide any future structure declarations with the same tag.

```
struct x f1;   /* becomes hidden */
```

Detection:

If the key word *struct* is detected and is followed by a token and a ; it is considered an incomplete structure declaration. Incomplete structure declarations will be flagged with id 11.

Correction:

To correct this incompatibility the automated system will simply remove the incomplete declaration entirely since it had no contribution to the program.

7.2.3.2 Inductive Type

The following incompatibility types below are inductive. This means that they can only be detected by the automated process. Corrections must be done manually.

7.2.3.2.1 Single/Double Precision Calculations

In Sun C the only actual floating-point type was **double**

(8 bytes). Thus, Sun C promotes the operands of floating point expressions to type **double**. Even in expressions involving only float, each float would always be converted to a **double**, the operations would be performed and the result converted back to **single** (4 bytes) precision. ANSI C allows operations on floats to be performed in **single** precision calculations. The effect of this is that a float stores about 6 decimal places of accuracy, and a double stores about 15. This would be a concern for applications that are precision intensive. The "custom made" applications internal to the MMDC involve inventory and cost quantity calculations thus, single precision calculations are probably not a problem.

Detection:

If a variable declared as a float resides inside an expression it will be flagged as an incompatibility with id 12. The software engineer running the automated system will have to evaluate the detection and determine what, if anything needs to be done to correct the incompatibility.

7.2.3.2.2 Lvalue Type Casting

The Sun C compiler would allow programmers to put casts on the left hand side of an assignment operator. This is no longer allowed in Sun ANSI C. Code will have to be rewritten to use an actual pointer variable that is previously declared to the type of the cast. For example:

```
int *var1;
```

```
(char *)var1 = &var2;
```

var1 is cast as a character pointer and for that instant is not an integer pointer. Var1 is assign the address of var2. Under ANSI C the declaration would need to be initially declared as character

```
char *var1;
```

and then assigned the address of var2

```
var1 = &var2;.
```

Detection:

The key elements in the detection of an lvalue type cast is to first determine if a cast is detected, then determine if the cast resides on the left hand side of an assignment operator (listed in section 8.2.3.1.4). Any code that contains lvalue casting will be flagged with id 13. Corrections to address this incompatibility will need to be done manually. An engineer will need to evaluate whether or not the datatype declaration of the variable can be changed to the cast value.

7.2.3.2.3 Types Allowed On Bitfields

An *int* or *unsigned* identifier can be declared to consist of a specified number of bits which identifies its width, typically they are members of a structure. Such a member is called a bit field.[6] The structure in Figure 25 below is an example of how you would declare a bitfield within a structure:

```
struct S
{
     unsigned a:4;
     unsigned  :7;  /* unnamed bitfield */
     unsigned b:5;
} st;
```

Figure 25. Bitfield Structure

Sun C allows bitfields of all integral types, including unnamed bitfields. Sun ANSI C supports bitfields only of the type *int, unsigned int,* and *signed int.* Other types are undefined. All the integral types are discussed below:

Signed integral types,

short int (2 bytes, 16 bits)

long int (4 bytes, 32 bits)

int (2 or 4 bytes, in Sun ANSI C it is 4 bytes)

The specifier *short int* is equivalent to *short* and the specifier *long int* is equivalent to *long.*[7] Unsigned integral types are declared the same as the signed just prefixed with the *unsigned* specifier. Finally, *char* is an integral type and can be used in integer expressions and is 1 byte or 8 bits long. The elimination of long as a legal bitfield type means that the longest bitfield that you can portably create is 16 bits, the minimum size of an *int.*

Detection:

If a an integral declaration type is detected and is

followed by a token and a colon and a semi-colon, a bitfield integral type has been detected. If the integral datatype is not *unsigned* or *int* the bitfield is undefined and is illegal. Detection of any illegal bitfield integral types will be flagged with id 14. The correction for this incompatibility will be done manually since it is not clear as to what the intent was for the integral type specified. Since *int* in Sun ANSI C is 4 bytes long the solution may be to change Sun C bitfield types of *long* or *short* to ANSI supported type of *int*.

7.2.3.2.4 Labels in Expressions

Sun C treats labels as (void *) lvalues. ANSI C does not allow labels in expressions.

Detection:

Detection of labels within an expression will be flagged with id 15. The correction will have to be invoked manually. The expression will have to be restated without the label. It will have to be determined weather a variable or constant value will take the place of the label.

7.2.3.2.5 Switch Condition Type

Sun C allows *float* and *double* types in a switch conditional by converting them to int. Switch statements in Sun C were evaluated solely as an *int*. The program in Figure 26 below,

```
main()
{
     float y = 4.3213;
     switch(y)
     {
          case 1:
               flag = 1;
               break;
          case 4:          /* Converted value of float */
               flag = 2;
               break;
          default
     }
]
```

Figure 26. Switch Statement.

would yield a compiler error in ANSI C. Only integral types *int, char,* and *enumerated* are evaluated for the switch condition type in ANSI C.

Detection:

If the identifier inside if a switch statement is not one of the three integral types stated above it be flagged with id 16. The key element in detection is to look for the key word switch and evaluate what integral type its identifier has. It is not clear as to the reason why the expression for a switch statement would not be of integral type. Thus correction for such an incompatibility would have be done manually.

7.2.3.2.6 Extern & Static Function Declarations Inside a Block

Sun C programs can contain several functions within a single file. In Sun C, variables that are declared as *extern* or *static* inside the individual functions were defined to the

scope of the file. This is not true in Sun ANSI C. There only defined to the scope of the function they were defined inside.

Detection:

For detection, any locally declared variables, variables declared within a function, that are preceded with *extern* or *static* will be flagged with id 17. Sun C promoted these function declarations to file scope. Under Sun ANSI C these variables need to be declared as static global types . This way the variables will stay promoted to file scope. This needs to be done manually to make sure that the global declaration will not conflict with existing global declarations.

7.2.3.2.7 Unsigned Preserving or Value Preserving

Sun C supports unsigned preserving. This means that *unsigned char* and *unsigned short* are converted to *unsigned int* when they need to be widened. ANSI C supports value preserving. This means that *unsigned char* and *unsigned short* are converted into *signed int* when the need to be widened, if that type can represent all the values of the original type; otherwise they are converted to *unsigned int*. In summary, in Sun C code when a signed type mixes with an unsigned type, the result is an unsigned type. In ANSI C when a signed type mixes with an unsigned type, the result is an signed type.

Detection:

If an expression contains values or identifiers that mix both signed and unsigned types it will be flagged with id 18 upon detection. It must be determined whether or not value preserving will effect the behavior of the program. *Unsigned* or *signed char* types are 8 bits wide. *Signed char* has a minimum value of -128 and a maximum value of 127. Unsigned char's min and max is 0 and 255 respectively. *Unsigned* or *signed short* types are 16 bits wide. *Signed short* min and max is -32768 and 32767 respectively. *Unsigned short* min and max values range is 0 and 65536.

7.2.3.2.8 Typedef Names In Formal Parameter Lists(1 pg 164)

Sun C allows one to use *typedef* names as formal parameter names in a function declaration and would hide the *typedef* declaration. ANSI C disallows the use of an identifier declared as a *typedef* name as a formal parameter. An example is shown in Figure 27 below:

```
typedef int * ptr; /* ptr is a typedef variable */
main()
{
     int d,e,f;
     /* statements */
     f(d,e,f);
}
f(a, ptr, c) /* typedef inside parameter list */
int a,ptr,c
{
     /* statements */
}
```

Figure 27. Typedef in Parameter List.

Detection:

Functions that contain formal parameters previously declared as typedefs will be flagged with id 19. Since these parameters could be used extensively within a function it could be a very tedious design process to code the correction of every detection of the parameter within the function. It would be more effective to manually correct this type of incompatibility.

7.2.3.2.9 Integer Constants

Sun C accepts 8 or 9 in octal escape sequences; ANSI C does not. Escape sequences are always prefixed with a back slash and represent a single character. Octal numbers are prefixed with a 0. Thus octal escape sequences will be prefixed with \0. The \ is called the escape character and is used to escape the usual meaning of the character that follows it. Octal escape sequence 8 is displayed as \08. Sun C would accept the character constant "\128" as equivalent to "\130".[8] The problem is that the first character constant has a different meaning now in ANSI C.

Detection:

Code that contains octal escape sequences that include 8 or 9 will be flagged with id 20. The correction will be done by the software engineer because the original values that escape sequences 8 and 9 contained are now unknown.

7.2.4 Data Flow Diagrams

Data Flow Diagrams (DFD's) are used for the detailed design phase of the automated system. One inductive and one deductive incompatibility type will be chosen from the deductive and inductive incompatibility descriptions defined in the previous section. A representative approach from both is shown in the data flow diagram design. Data Flow Diagrams (DFD's) provide a graphical approach for describing the information flow characteristics within a system.[9] DFD's are used as a mechanism for creating a top level structural design for software. The objective of the structure design is to develop the smallest number of distinct modules that satisfy functional requirements.[10]

7.2.4.1 General Design Approach

Data Flow Diagrams have the following basic characteristics:

1. Information (data flow) is represented by labeled arrows.
2. Transformation (processes) are represented by labeled circles.
3. A Database log or retrieval is represented by a labelled, double line.

The fundamental model, shown in Figure 28, is refined through a series of DFD's until sufficient detail is developed. The information into and out of a transform remains fixed

regardless of the level of refinement.

7.2.4.2 Requirements and Design

From the requirements section the automated process is to convert an entire "custom made" application at a time. All the source files, .c files, that encompass an entire application is found within the Makefile. All the "custom made" applications within the MMDC have a Makefile. Thus, the input to the system must be the Makefile path. The system will retrieve all the files that make up an application from the Makefile. Database requirements state that the automated system must keep track of the line count while checking for incompatibilities. Any time the system detects an incompatibility, information of the type is logged to the database. The system must log the time it takes 6625 lines of code to be checked for all 20 possible incompatibilities. Once an entire "custom made" application is converted the system must log the total number of incompatibilities detected to the database. The system must provide the capability to print out a test sheet that displays the logged database information for performance testing purposes. Up to this point all of the requirements pertain to both inductive and deductive types. The only additional requirement of the inductive incompatibilities is that the engineer must have the capability to edit files to make corrections when inductive type of incompatibilities are detected. Thus the location of

the incompatibility detection within the file needs to be displayed to the screen. The editing mechanism that is incorporated into the design is the use of Unix text editors. Unix text editors are what is used by the software engineering group to develop software. At the Unix prompt the command "textedit" will bring up a test editor. Sun C supports simple system calls that allow such commands to be activated.

7.2.4.3 Assignment Operator & Lvalue Type Casting

The assignment operator and the lvalue type casting are the two incompatibility types that are chosen for DFD design. Much of the DFD design process is not any different between the two types. The .c file retrieval process, logging database information, generating the "print status sheet", generating next .c and current incompatibility id's and finally compile and run capability is the same whether your dealing with inductive or deductive types. The correction process for inductive incompatibility types are all the same. The engineer will actually be making the correction so the automated system just has to support a way for editing each time an inductive type is detected. DFD differences between inductive and deductive types is revealed in how each one is detected and how each deductive type is corrected. All the id's that uniquely identify an incompatibility are hard coded into a global array variable and retrieved as needed. Unique aspects about the two types chosen are discussed below.

Assignment Operator (Deductive Type):

The key incompatibility problem here is that Sun ANSI C treats assignment operators as single tokens thus not allowing white space between the operator pairs. It is pertinent for detection purposes that these operator pairs are hard coded into a global array as part of an initial declaration task. The array will then be known to all the files of the system. The assignment operators are listed in section 8.2.3.1.4 above. The DfD's utilize the array in the detection process to help verify an incompatibility detection.

Lvalue Type Casting(Inductive Type):

ANSI C will not support the type casting of lvalues. The initial task to support detection of this incompatibility type is to hard code all the different datatypes into a global array. The DfD's utilize the array in the detection process to help verify an incompatibility detection. The software engineer will make the correction. It is probable that the lvalue identifier that is type casted is referenced more that once throughout the "custom made" application. The DFD's on the next several pages establish the software design data flow of the system. Modules are refined down until they represent functions.

The Interactive convert Software executable that will convert Sun C code to ANSI C will initially need , upon start-up, the Makefile path. The final output that the system will provide is a "Performance Status Sheet", to aid in the testing of performance requirements, and most importantly the system will have converted the Sun C code of the specified application into Sun ANSI C. The figure below is the first level DFD of the system.

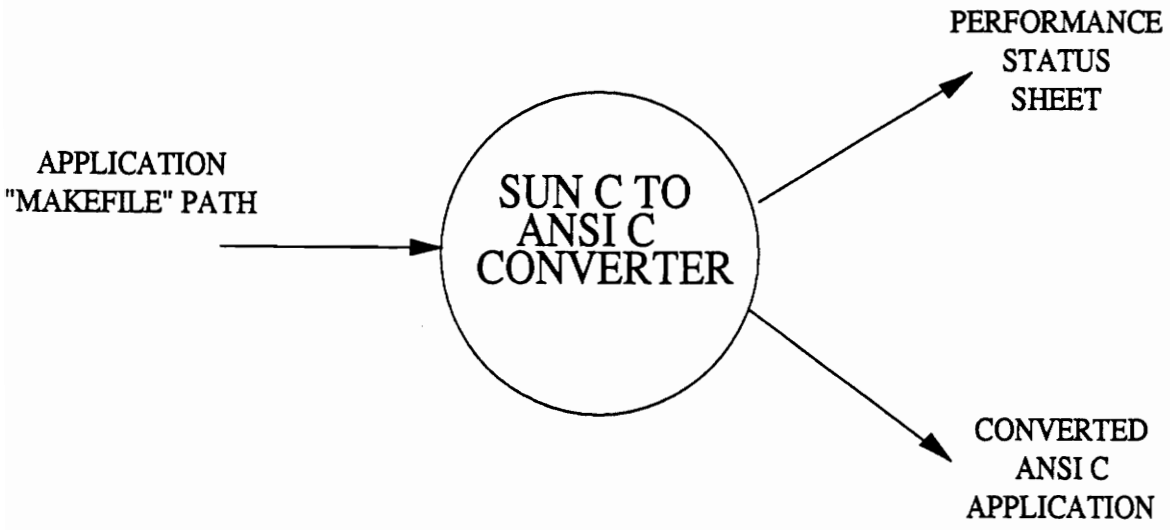


FIGURE 28. "CONVERTER" LEVEL 1.

There are seven major functions that the system will need to perform. From the Makefile the converter software will need to retrieve all the .c files. Once the .c files are retrieved and stored into an array each of the lines of code within each .c file, retrieved one at a time from the array, will need to be checked for potential "Sun C ANSI C" incompatibilities. The system will need to know when to generate the next file and incompatibility type (explained later). The id of each incompatibility type will be stored in an array. Each incompatibility will be tied to an id value anywhere from 1 to 20. The system will Log information into the three table database each time an incompatibility detection has been made, each time 6625 lines of code have been successfully checked for all 20 possible incompatibility types and finally when the conversion of the application has completed. Upon completion of the conversion the engineer may opt to print out a "Print Status Sheet". The system will also allow the engineer to compile and run the converted source code with the ANSI C compiler to determine if the converted application was successfully converted. The Figure X on the following page is for inductive type incompatibilities. The system will display information to the screen when an inductive type of incompatibility has been detected. In order for the engineer to correct these types the system will allow the engineer to bring up a text editor to make the correction.

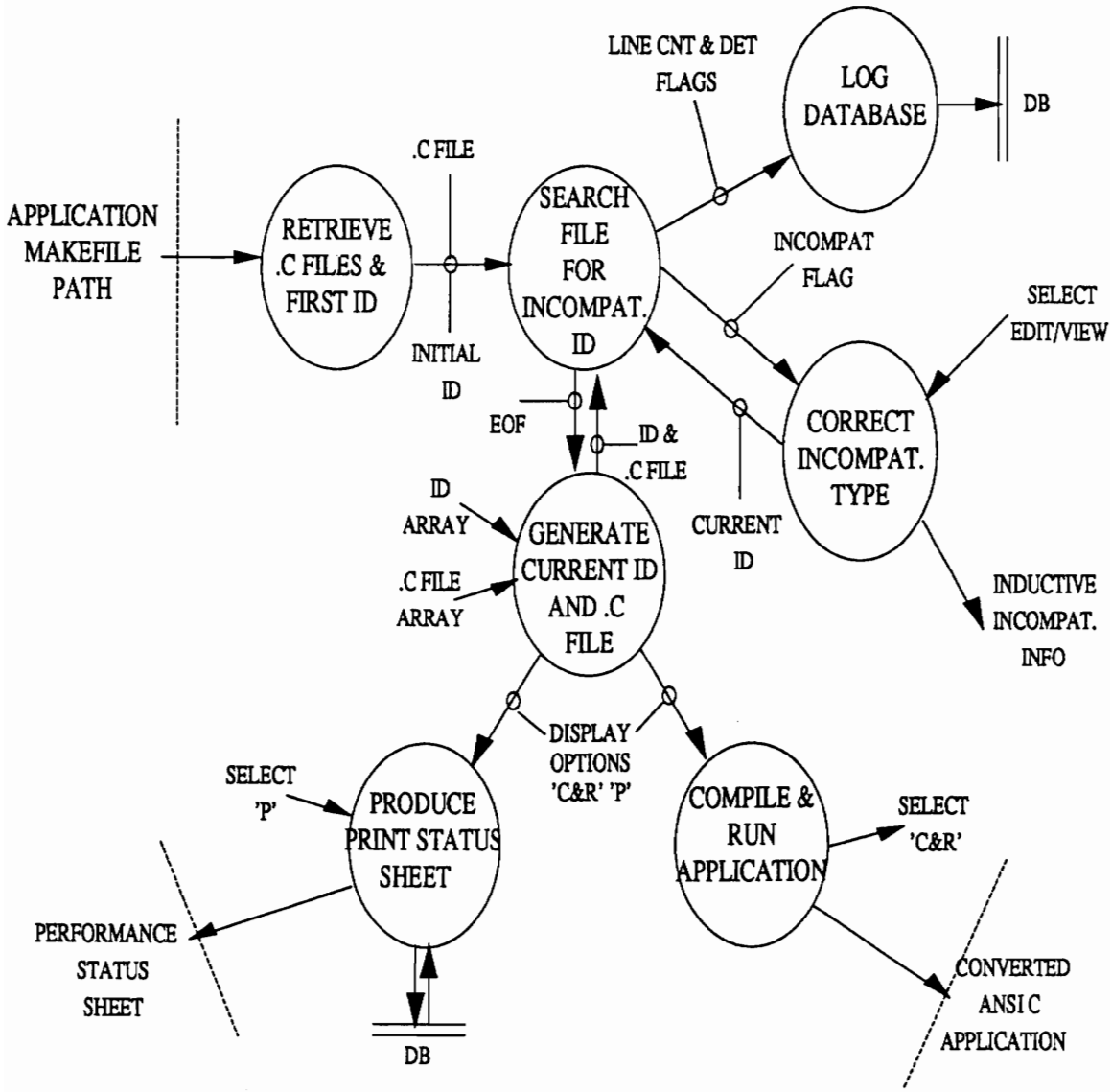


FIGURE 29. "CONVERTER" LEVEL 2 (INDUCTIVE).

The system will function the same for the deductive types of incompatibilities except for when a deductive incompatibility type is detected, it will not display anything to the screen. The system will just proceed to correct the incompatibility. Thus, the engineer will not need to bring up a text editor to make corrections. Note, in the figure below, the "inductive incompat info" and "select edit/view" data flow arrows have been removed from the "CORRECT INCOMPAT TYPE" module.

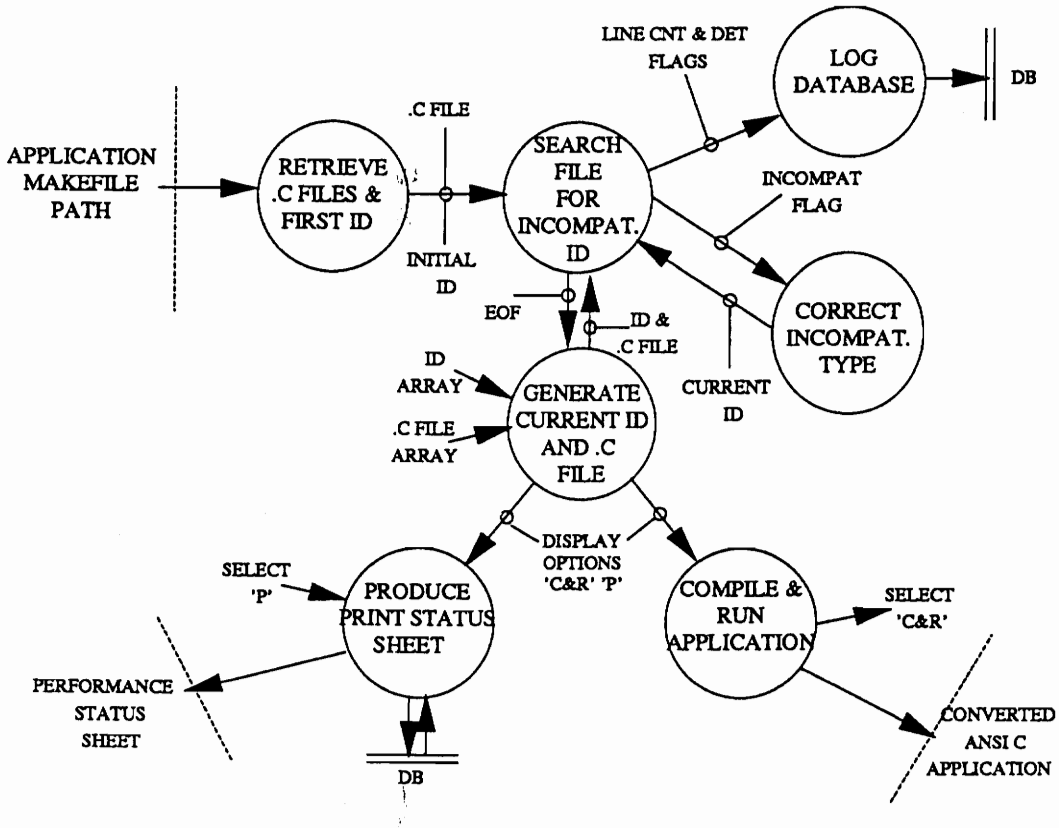


FIGURE 30. "CONVERTER" LEVEL 2 (DEDUCTIVE).

The DFD, in

figure below, expands the functionality of the level 2 "RETRIEVE .C FILES & FIRST ID" module into its sub functions. Given the Makefile path, the system will locate the Makefile and load up a start time variable and the id variable with the first id in the id array. Once the Makefile is found the file will be opened and traversed. Every time a .c file is detected the "OPEN FILE LOAD UP .C FILE ARRAY" module will inset the name of the file into the .c array. The .c array is then passed to the "GET FIRST .C File" module. This module will simply retrieve the fist file from the array that is to be examined.

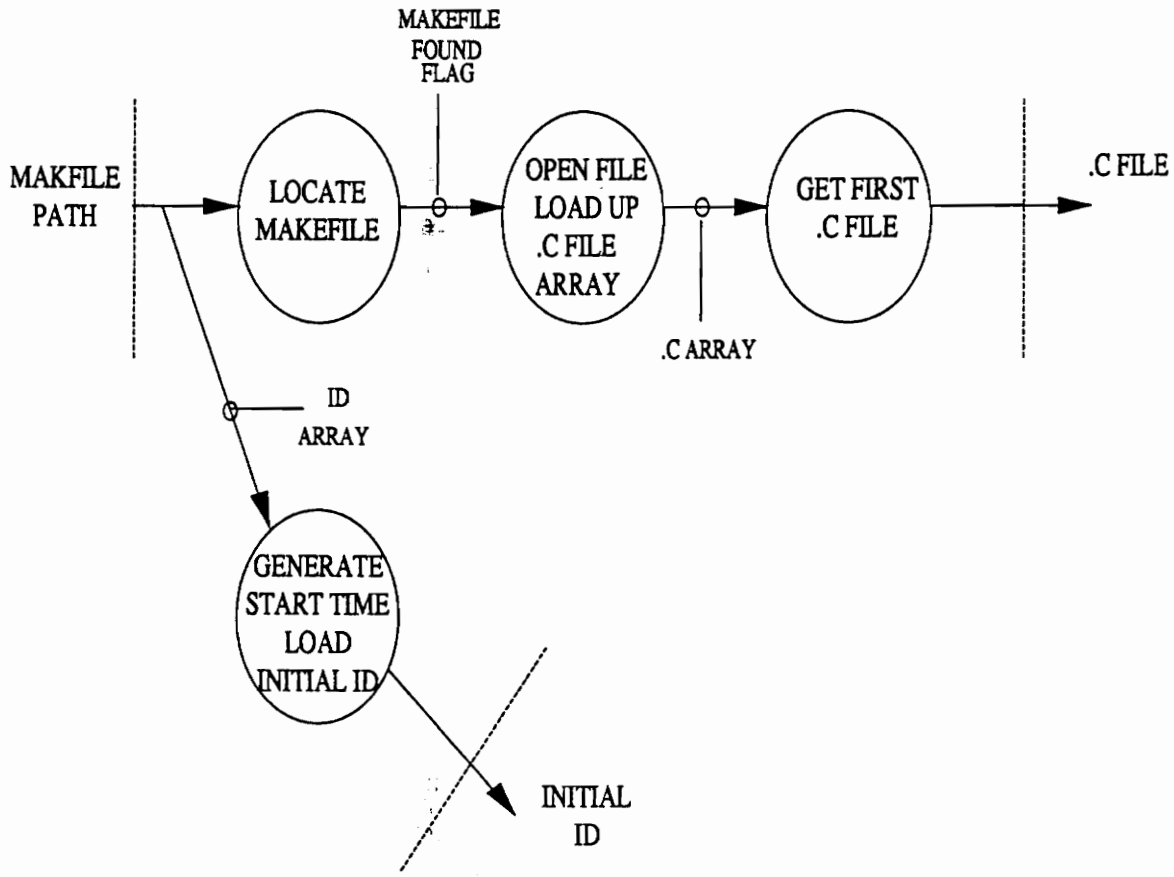


FIGURE 31. "CONVERTER" LEVEL 3 (.C FILES).

The DFD, in the figure below, expands the functionality of the level 2 (inductive type) "SEARCH FILE FOR INCOMPAT ID" module into its sub functions. The function described is for the inductive incompatibility with id 13, "Lvalue Type Casting". The .C file is passed to the "CHECK EACH LINE FOR CASTING & INC COUNTER" module (elaborated on later). When the whole file is traversed an End Of File (EOF) flag is set for use by the Level 2 "GENERATE CURRENT ID AND .C FILE" module. Each time a line is traversed the system must check if line count 6625 has been reached and if current id is 20. If so the "Line Cnt" flag will need to be set. This flag tells the system that information will need to be logged to the database. If a type cast is detected the system must determine if it is an Lvalue cast. If so the "Incompat detected flag" is set.

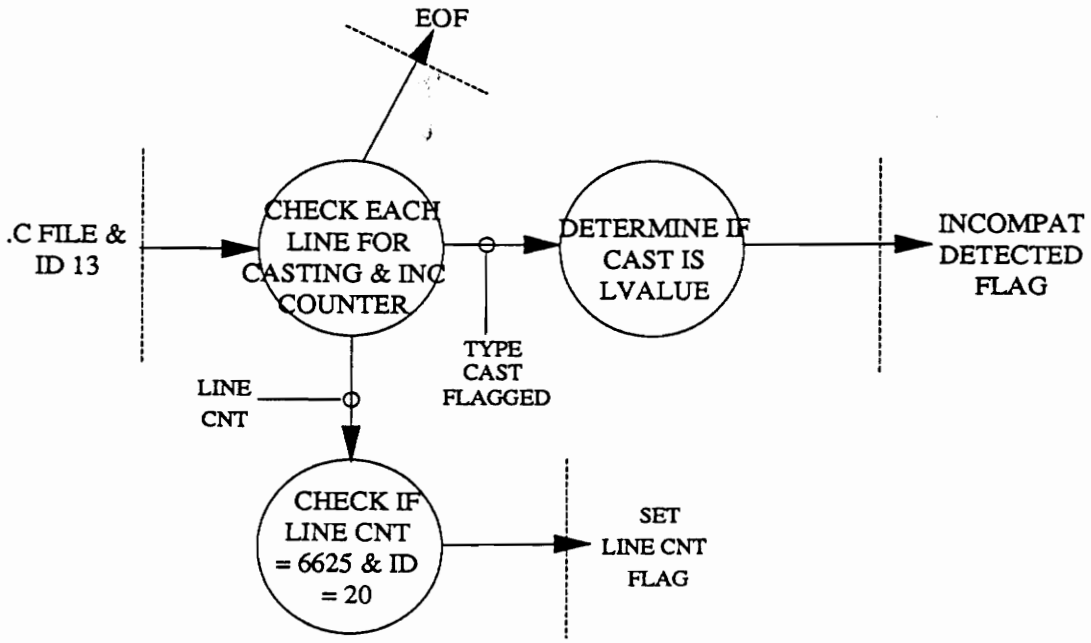


FIGURE 32. "CONVERTER" LEVEL 3 (INDUCTIVE DETECTION).

The DFD, in the figure below, expands the functionality of the level 3 (inductive type) "CHECK EACH LINE FOR CASTING & INC COUNTER" module into its sub functions. First the system will need to identify all the declared identifiers throughout the file. Once the identifier array is established each use of each identifier, within the array will need to be found throughout the file. Once found, an additional check will be made to determine if the identifier is prefixed by a type cast.

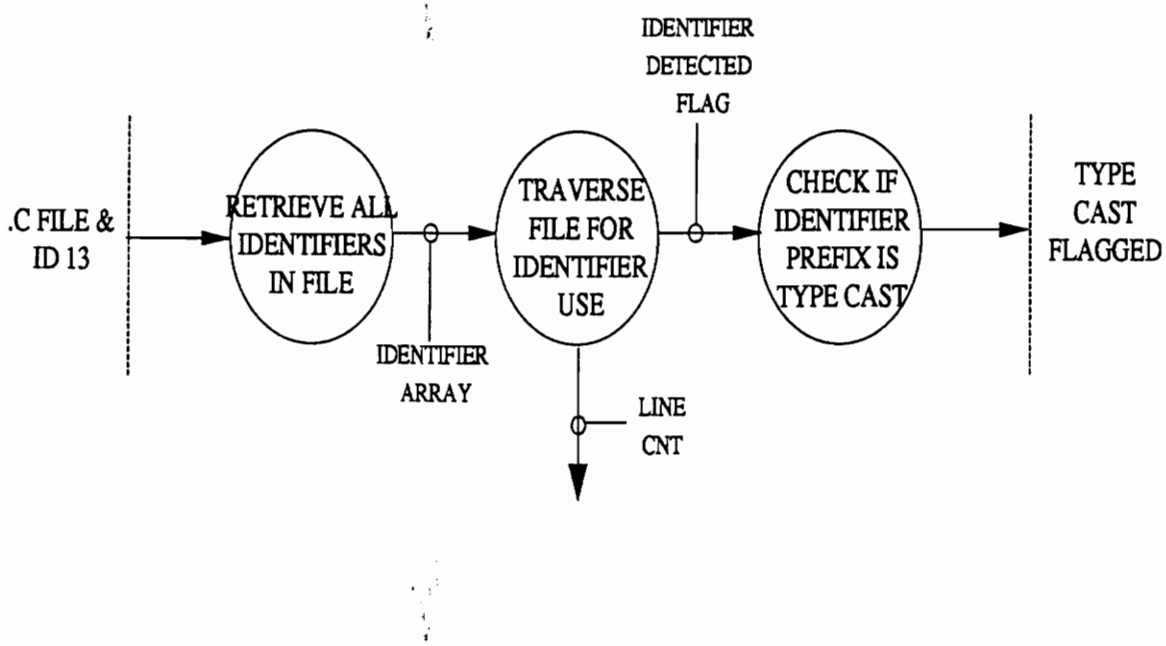


FIGURE 33. "CONVERTER" LEVEL 4 (INDUCTIVE DETECTION).

The DFD, in the figure below, expands the functionality of the level 3 (inductive type) "DETERMINE IF CAST IS LVALUE" module into its sub functions. A Token is anything surrounded by a white space. Once an indentifier is determined to have a type cast, the token to the right needs to be evaluated to determine if it is an assignment operator. An array of assignment operators will have been initially set up. The token is compared to all values in array. If token matches an assignment operator then set A.O flag. If set an incompatibility is detected.

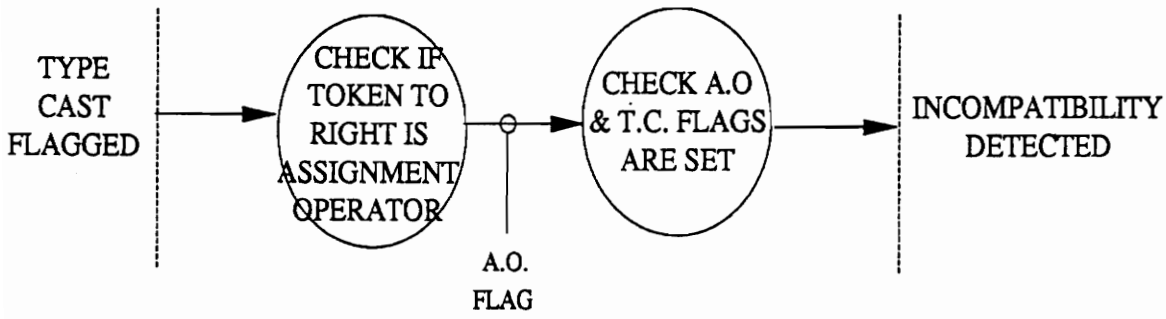


FIGURE 34. "CONVERTER" LEVEL 4. (INDUCTIVE DETECTION).

The DFD, in the figure below, expands the functionality of the level 3 (inductive type) "DETERMINE IF CAST IS LVALUE" module into its sub functions. A Token is anything surrounded by a white space. Once an indentifier is determined to have a type cast, the token to the right needs to be evaluated to determine if it is an assignment operator. An array of assignment operators will have been initially set up. The token is compared to all values in array. If token matches an assignment operator then set A.O flag. If set an incompatibility is detected.

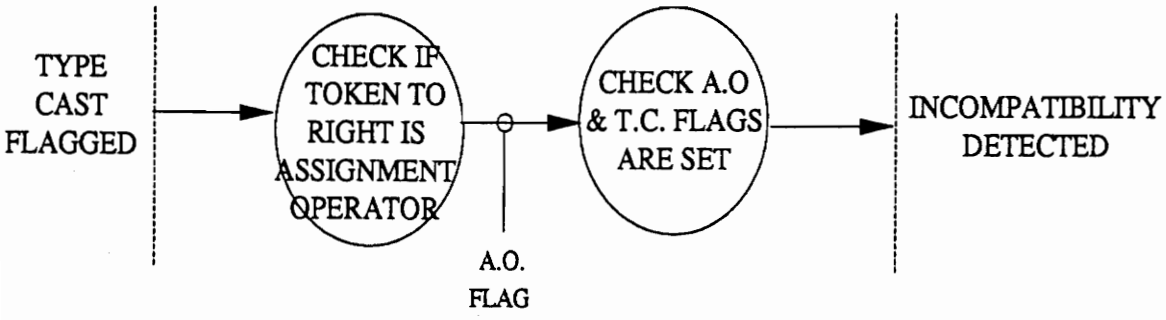


FIGURE 34. "CONVERTER" LEVEL 4. (INDUCTIVE DETECTION).

The DFD, in the figure below, expands the functionality of the level 2 (inductive type) "CORRECT INCOMPAT TYPE" module into its sub functions. All the inductive incompatibility types will be corrected by the system the same way. Upon detection of an incompatibility the "GENERATE INDUCTIVE INCOMPAT INFO & OP" module will check if current id is inductive type if so it will display to the engineer info needed to make correction for incompatibility type. The engineer will can opt to edit the file where incompatibility was detected or prompt system to continue. The file will be displayed via a text editor. The system will stay in a wait state until the text editor process has exited. The system will then continue on its search to detect incompatibility types of the current id type

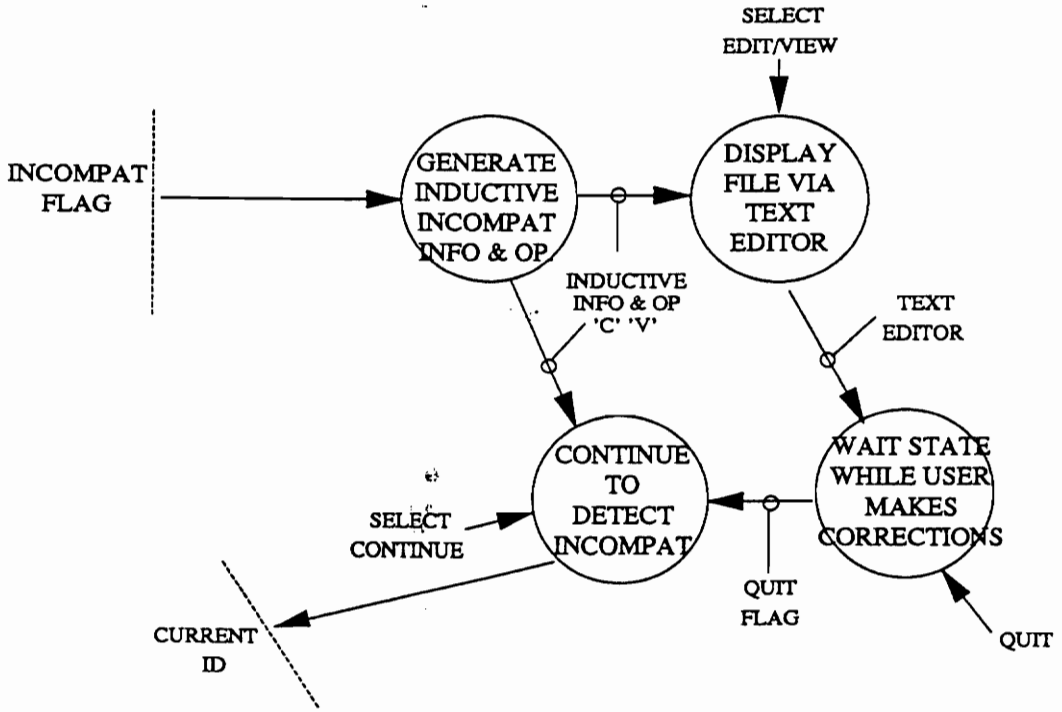


FIGURE 35. "CONVERTER" LEVEL 3 (INDUCTIVE CORRECTION).

The DFD, in the figure below, expands the functionality of the level 2 (deductive type) "SEARCH FILE FOR INCOMPAT ID" module into its sub functions. The "Assignment Operator" (id 4) incompatibility type is the deductive incompatibility type looked at for the system. The current .c file will be traversed for possible detection of id 4. The globally initialized assignment operator array will be used to determine if a Sun C version of the assignment operator has been detected. Each time a line is traversed the system must check if line count 6625 has been reached and if current id is 20. If so the "Line Cnt" flag will need to be set. This flag tells the system that information will need to be logged to the database. If an assignment operator is detected and it contains white space, an incompatibility is then detected.

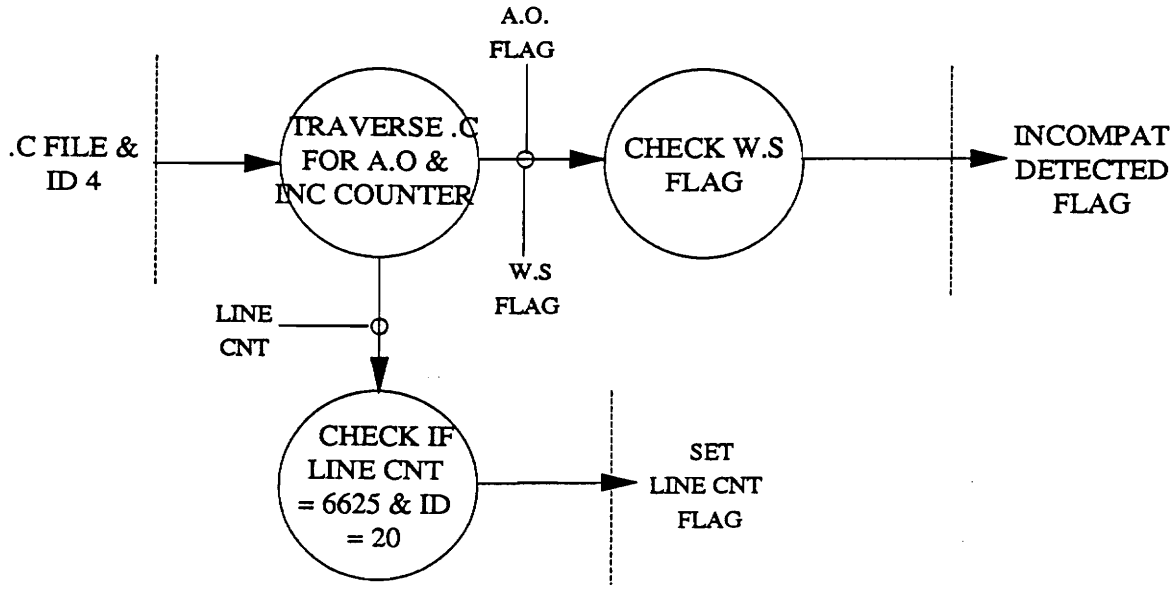


FIGURE 36. "CONVERTER" LEVEL 3 (DEDUCTIVE DETECTION).

The DFD, in the figure below, expands the functionality of the level 3 (deductive type) "TRAVERSE .C FOR A.O. & INC COUNTER" module into its sub functions. Each line of the file is read and each token retrieved. The assignment operator array is checked to see if the token retrieved is an exact match if so white space flag remains unset. If the token is part of an assignment operator then there is a potential for a possible detection. Set the A.O. flag and then check and see if second token is "=". If so then set assignment operator flag and white space flag.

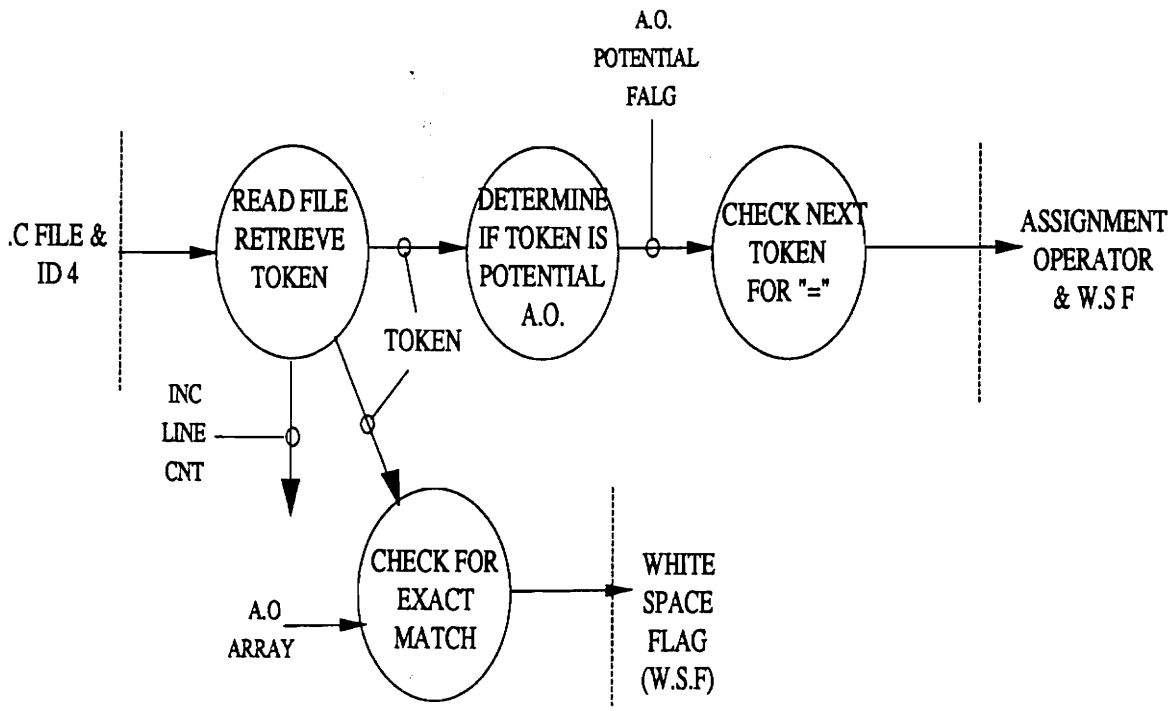


FIGURE 37. "CONVERTER" LEVEL 4 (DEDUCTIVE DETECTION).

The DFD, in the figure below, expands the functionality of the level 2 (deductive type) "CORRECT INCOMPAT TYPE" module into its sub functions. The white space is removed from the assignment operator variable. The new token is written back to the file is the same spot as the old token was found. The system will continue to detect next incompatibility.

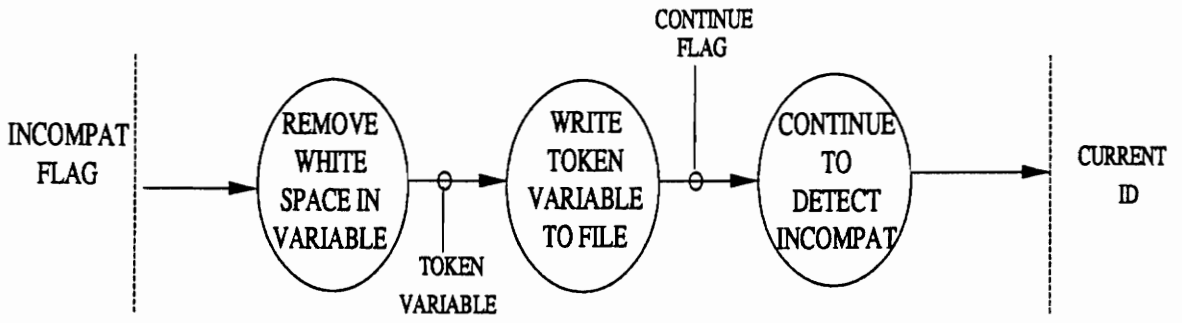


FIGURE 38. "CONVERTER" LEVEL 3 (DEDUCTIVE CORRECTION).

The DFD, in the figure below, expands the functionality of the level 2 "LOG DATABASE" module into its sub functions. If detection flag is set table 2 info is logged. If line count is 6625 then table 1 info is logged.

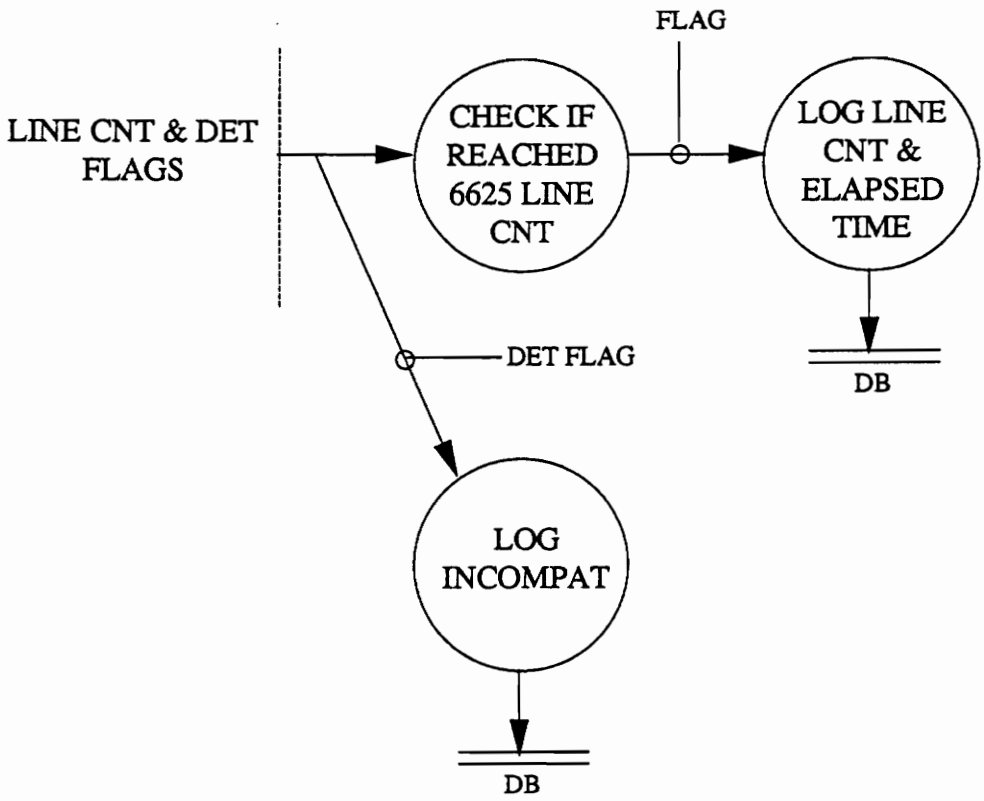


FIGURE 39. "CONVERTER" LEVEL 3 (DATABASE LOG).

The DFD, in the figure below, expands the functionality of the level 2 "GENERATE CURRENT ID AC .C FILE" module into its sub functions. If the end of file is reached the system will get the next id type. If new id is 1, meaning the array was completed, then the next .c must be retrieved. If the current .c file is the last then set flag. The system will then generate the conversion complete message and the incompatibility totals will be logged to the database. The system will also display the options to allow the engineers to compile and run the newly converted application and print out a "Performance Status Sheet".

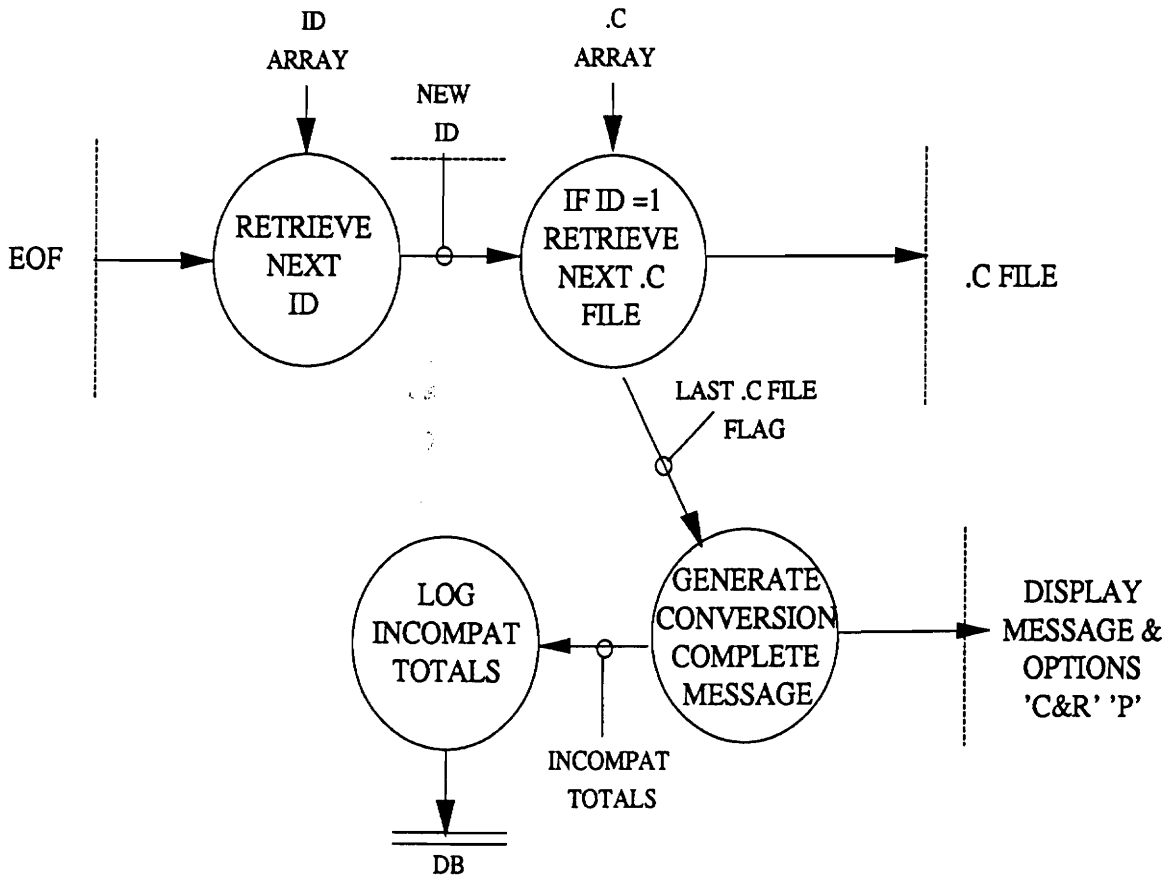


FIGURE 40. "CONVERTER" LEVEL 3 (ID & .C GENERATOR).

The DFD, in the figure below, expands the functionality of the level 2 "PRODUCE PRINT STATUS SHEET" module into its sub functions. The generation of this sheet will be totally dependant on the information logged into the database. The info from the table 2 will be retrieved and stored into a link list. The final totals from table 3 will be stored in a variable. The information from the variable and the link list will be displayed on the "Performance Status Sheet" in a formatted fashion.

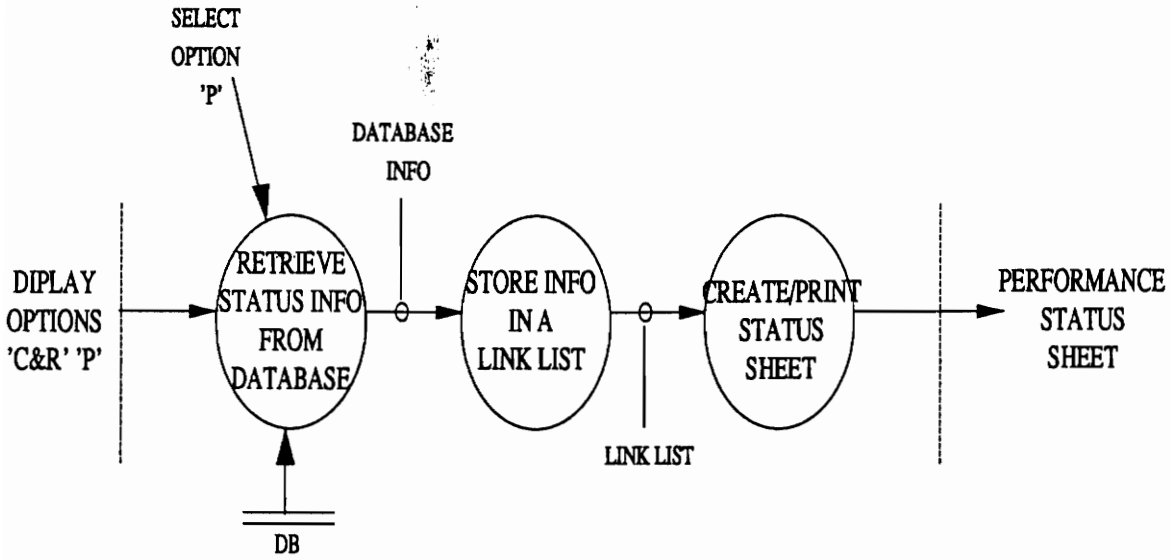


FIGURE 41. "CONVERTER" LEVEL 3 (PRINT OPTION).

The DFD, in the figure below, expands the functionality of the level 2 "COMPILE & RUN APPLICATION" module into its sub functions. The ANSI C compile command can be generated and issued with system calls. Upon failure of the compile the system will exit with a compilation failed error. If application compiles successfully then the system will generate the application run command then issue the command.

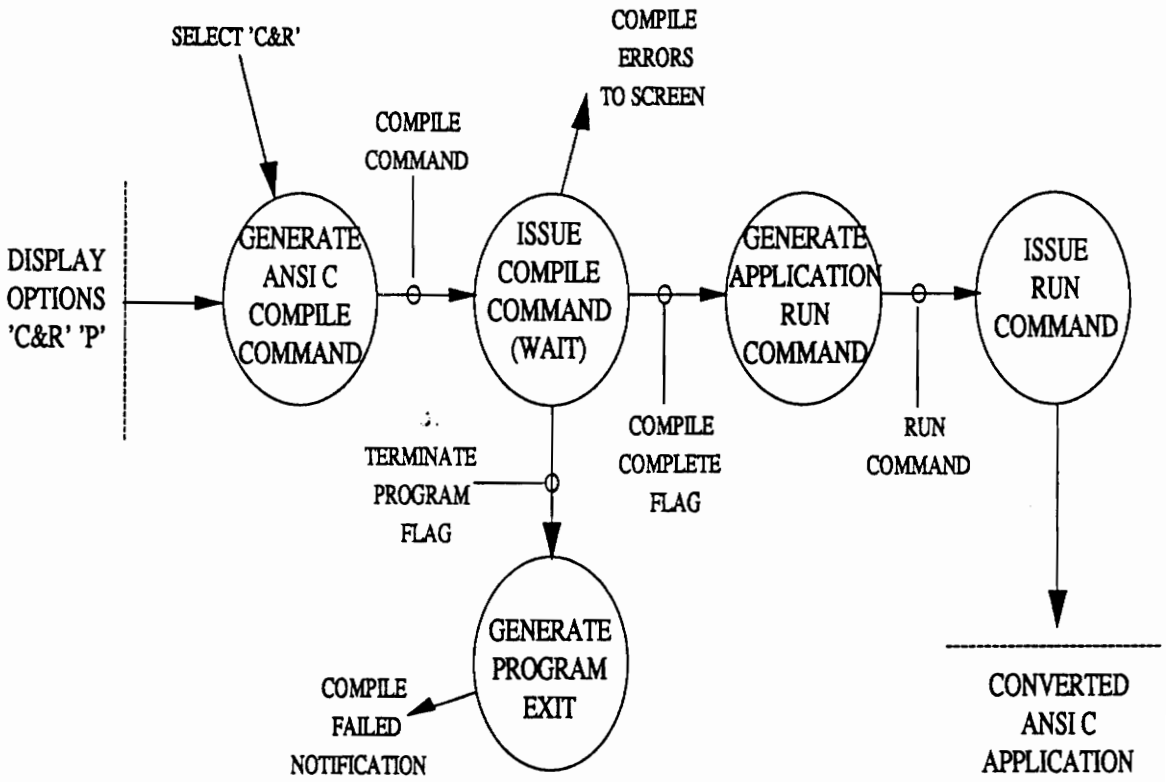


FIGURE 42. "CONVERTER" LEVEL 3 (COMPILE & RUN).

8.0 Test System

8.1 Selection of Test Items and Test Site

The testing will occur within the MMDC utilizing the operational LAN. Prototype applications that will be used for testing the automated system will be a copy of any one of the six "custom made" applications that currently are used. A copy of any of the applications can be made at any time for testing.

8.2 Test Personnel and Training

Individuals that will be doing the testing will be two software test engineers since they have a major role in the operation and maintenance of the automated system. These engineers will have been trained on the operational functions of the system software since they will have taken part in its development.

8.3 Test and Support Equipment

Hardware test equipment include use of a Sparc 2 workstation, printer and a F/S disk drive. Software support equipment will be the operating system software of the MMDC LAN, the automated system software and an ANSI C compiler.

8.4 Test Supply Support

All hardware spares are stored within Logistics. HMO may replace any support hardware that fails during testing with the appropriate spare part stored.

8.5 Reliability Testing

8.5.1 Test MTBF requirement

1. Utilize a copy of one of the operational applications as a prototype for testing.
2. Utilize the automated process to convert the entire application.
3. Compile converted application with the ANSI C compiler to test for compile time errors that were not detected property by the automated system.
4. Run application and test for run time errors that may exist.
5. Since failures will occur randomly the number of all errors detected are averaged over the amount of time it took to convert the application. From this a MTBF is estimated.
6. Acceptance of test will depend on if failures fall within the stated requirement value give or take .2hrs.
7. Redesign and retesting must occur until desired value is met.

8.6 Performance Testing

8.6.1 Test for 98% accuracy

1. Utilize a copy of one of the operational applications as a prototype for testing.
2. Convert an entire application.
3. Test personnel must print out the Performance Status

Sheet, sample shown in Figure 43. From the sheet the total number of incompatibilities detected is revealed.

4. Compile and run the converted application. The number of failures of the automated process to detect/correct properly will be compared to the total number of incompatibilities that existed within the converted application. The percent accuracy can then be determined.

$$\%accuracy = (total-fail)/(total) \times 100 \quad \text{(Eq. 15)}$$

5. For every 100 errors detected 98 need to have been detected and corrected properly.

8.6.2 Test for code conversion rate of 6625 per day

1. Perform steps 1 and 2 stated above.
2. Test personnel must print out the Performance Status Sheet.

   From the sheet the test personnel can determine if 6625 lines of code is converted every 8 hours.
3. Redesign and re-test conversion process until the above requirement is satisfied.

8.7 Maintainability Testing

8.7.1 Test for MCT of 6 minutes

1. Perform steps 1, 2 and 3 of the MTBF requirement.
2. Record the average corrective maintenance time it takes to make corrections to automated system for

correction/detection errors.

3. If the average MCT of software repairs exceed 6 minutes by more than 4 minutes, the Mct requirement will need to be changed to a more realistic time. This will in turn change the operational life cycle of the system since as entire applications are converted failures in the conversion process will need to be corrected at the new Mct.

4. Before making any recommendations to increase Mct requirement remember that any fixes to the code during test time should not reoccur when the system is in operational use. Thus if there are one or two long corrective times for repairs to code, that significantly increased the Mct value, and the rest are well within the 6 minute requirement, a recommendation to change the Mct requirement should not be made.

8.8 Test of Converted Applications

After applications have been converted and compiled with the ANSI C compiler the test group engineers must test every aspect of the converted application to make sure it functions properly. Original test procedures that currently exist for the "custom made" applications will need to be looked at to test the functionality of the "custom made" applications that have been converted.

PERFORMANCE STATUS PRINTOUT

Application Name: Sample

Application Conversion Rate Statistics

Incremental Date	Cumulative Line Count	Elapsed Time
04/01/96	6625	8 hrs
04/02/96	13250	7.8 hrs
04/03/96	19875	8.5 hrs
.	.	.
.	.	.
.	.	.
4/10/96	66250	7.5

Final Conversion Totals

Total # of Incompatibilities Detected	Inductive Count	Deductive Count
4416	2013	2403

Figure 43 Performance Status Sheet (Sample)

9.0 System Retirement

Once the "custom made" applications have been converted the system is to be kept accessible to the software engineering group, via the LAN, for one year. If over the course of the year it is determined that a "custom made" application failed due to a Sun C to ANSI C incompatibility error, corrections may be made to the system and the applications reconverted to fix all possible occurrences of the error that may exist. Once the year is up and no failures have occurred, the system may be archived to tape to free up disk space.

10.0 Conclusions

It was determined that it is not feasible to continue to stay with the current operating system on the MMDC LAN. The operating system will not preserve the current functionality needed on the LAN after mid July 1996. The "custom made" applications are dependant, in part, on proprietary libraries provided by Sun. The licensing of these libraries coincide with annual maintenance support, which goes away in mid July 1996. It is, however, feasible to upgrade the existing operating system to Solaris 2.2 and still preserve all of the existing functional requirements on the LAN. In order to accomplish this, a feasible conversion approach was developed that converted existing K&R C code to ANSI C to allow the existing "custom made" applications to function properly under Solaris 2.2.

A manual approach and an interactive automated system approach were looked at as methods to convert Sun C code to ANSI C. It was determined that there currently is no existing commercial package that resolves Sun C to ANSI C incompatibility differences. An interactive automated approach was the most feasible. The cost of the manual approach was $193,500 and the interactive automated approach cost was $79,400. Not only is the interactive automated approach the most cost effective but is also the approach that would be the most accurate in consistent detection. A manual

approach for a conversion of this magnitude allows for inconsistent detection and correction behavior whereas the automated approach is programmed for consistency.

The scope of the automated system pertains only to the MMDC "custom made" applications. The design of the system is specific to certain MMDC Makefile structuring and is compiled under the SunOS (unix based) operating system. The system is was not developed for any other types of platforms. Thus the system design would not be considered generic.

11.0 Recommendations

A study needs to be done that will determine what COTS packages are available that could replace the existing 6 "custom made" applications. Implementing COTS packages will alleviate some of the control the vendor has on the MMDC.

After the existing Sun C to ANSI C incompatibility differences have been resolved the "automated interactive system" should be expanded to address the prototyping differences. The prototyping differences that exist between Sun C and ANSI C are still supported under the ANSI C compiler but ANSI C compilers may eliminate the support of the old style, Sun C, prototypes in the future.

The design and development of the system should start in early May 1995 in order to meet the desired need date of July 1996 (when operating support of the LAN goes away). Data flow diagrams will need to be developed for the rest of the incompatibility types as part of the formal system design process. The representative approach can be used as a base model for system software design.

The "automated interactive system" should stay in circulation for at least a year after all the applications have been converted. If any failure occurs within any of the applications that have been converted and the failure can be attributed to Sun C and ANSI C incompatibility differences that were overlooked, the system can then be expanded to

address the failure.

References

[1] B.S. Blanchard and W. J. Fabrycky, Systems Engineering and Analysis, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1990, Page 57.

[2] Ken Arnold and John Peyton, A C User's Guide to ANSI C, New York:, Addison-Wesley Publishing Company, Inc., 1992, Page 30.

[3] Al Kelly and Ira Pohl, A Book on C (ANSI C), 2nd ed, Redwood City, Ca: Benjamin/Cummings, Inc., 1990, Page 72.

[4] A C User's Guide to ANSI C, Page 11.

[5] A C User's Guide to ANSI C, Page 23.

[6] A Book on C (ANSI C), Page 321.

[7] Samuel P. Harbison and Guy L. Steele Jr.,C: A Reference Manual, 2nd ed, New Jersey: Prentice-Hall, Inc., 1987, Page 83.

[8] A C User's Guide to ANSI C, Page 14.

[9] By the Staff of General Electric Company, Software Engineering Handbook, Bridgeport, Conn., McGraw-Hill, Inc., 1986, Page 4-6.

[10] Software Engineering Handbook, Page 5.