SOFTWARE MANUFACTURING

L. Bernstein
C. M. Yuhas*
Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

This paper suggests a unique method of organizing and staffing for the production of deliverable computer-based systems which takes advantage of assembly-line techniques. Although these production activities are common to most projects, Software Manufacturing is defined here as an inline, rather than support, function requiring special skills. A system development cycle is described, detailing the Software Manufacturing tasks with considerations for introducing this functions into existing development efforts. This organizational approach is concluded to lead to systematizing the production of software, to opening career opportunities for technician and production level people, and to better managed product development.

Introduction

When we consider the functions that must be performed in creating and delivering a large-scale computer-based system, the issue is fraught with difficulty. Structured programming, top-down design and other fashionable buzz-words immediately spring to mind. The problem is confounded by the bias of creative designers and managers, which was neatly put by Bertrand Russell when he said:

> "Work is of two kinds: first, altering the position of matter at or near the earth's surface relative to other such matter; second, telling other people to do so. The first kind is unpleasant and ill paid; the second is pleasant and highly paid."

Those very necessary functions of documentation, change control, and actual software production tend to be viewed by designers and managers as falling into the first category of work. Therefore, these functions are often relegated to clerks who are ill-suited to perform them, or they are done haphazardly at the expense of the design work and the customer, or they are not done at all.

This paper will address only those functions which get the product out the door and keep it working. That doing these functions efficiently requires an unique organization and a different "mental set" from the design processes will be discussed. We offer no rigid formulations, simply a viewpoint that has evolved from the experience of working with computer systems.

What Is Computer Software Manufacturing?

Our objective is to create a computer-based system, package it, deliver it to one or more customer sites, and keep it working. Organizing software development along the lines of using a chief programmer with the aid of librarians and assistant programmers is a notion calculated to improve innovation and productivity within the present concept of programming as an art. This is well and good for the "create" part of the objective, but who shall handle the remainder? Clerks are traditionally concerned with keeping established information orderly. The key word is "established." Figure 1 shows a program which could compute the largest value of 3 numbers in some language, on some machine. This is not a piece of "established" software because it is not properly named, nor is it formatted for subsequent update and inclusion in a data system. Since a computer program is not deliverable software by itself, it therefore cannot be adequately handled by clerks alone. We would suggest that the missing link which satisfies "package-deliver-keep it working" parts of the objective is the function of Software Manufacturing. We would also suggest that this function requires disciplines and organization alien to both the innovative designer and the traditional clerk. It is in the direct line of delivering the product and is not a support function. Given certain piece parts, the Software Manufacturing function produces a tangible product. The piece parts are as follows:

- initial code (programs, data descriptions, cataglogue procedures)
- initial documentation (design and user documentation)
- updates or changes to the initial code and documentation
- trouble reports
- configuration definitions.

The product is the deliverable software for each site, properly named and structured for configuration management purposes. The deliverable software is comprised of the following:

- computer programs
- data base descriptions
- job control language statements
- inventory listings
- user documentation.

What Skills Does It Take To Do Software Manufacturing?

Before we can intelligently discuss the skills we would like in a Software Manufacturing organization, we need to examine the tasks which must be done in getting from the piece parts to the assembled product. Though other tasks can be identified, the following are probably the minimum that must be done to get from here to there:

- accept individual components from designers
- update source code
- perform quality control on software format standards
- assemble object or executable code
- maintain listings
- build system for use by test team
- install on target machine
- check that documents and program releases exist together where needed
- keep track of troubles
- keep track of changes
- identify configuration of each system build and note changes
- prepare software for shipment
- reproduce software for each site
- ship software to sites with inventory lists.

Concurrently, someone must worry about purchasing computer time, issuing management reports, and all the other activities that keep the production wheels oiled.

Even a cursory glance at these tasks suggests that these activities can impede the actual design function if not done properly, yet they are too complex for traditional clerical levels. The process of building

*Former Bell Laboratories Employee

software and controlling it demands a technician with some software training. Skills in Job Control Languages, data base control and machine scheduling are needed. Additionally, to develop tracking, testing and report production subsystems, system analysis skills are necessary.

Finally, volume production and quality control demand an assembly-line, product-oriented frame of reference, with emphasis on scrupulous adherence to procedures and inventory control.

How May The Software Manufacturing Function Be Organized?

A system development cycle is shown in Figure 2, with the Software Manufacturing tasks introduced at two points. Between the programming group, doing the coding and module testing, and the system test group, doing the integration of modules, we find the block of Software Manufacturing tasks concerned with controlling and building a system for use by the test team. At the completion of integration and test are the Software Manufacturing tasks concerned with preparing and shipping the system. Top down design and structured programming technology may be applied during the definition, design and implementation phases of this cycle. Figure 3 shows a schematic of the manufacturing cycle. A "new unit" is anything we can label and keep track of as it is added to the system: new programs, new documents. The "update report" contains the change to either code or documents. A labelling convention identifies the location of the change. New units are placed in a data base. If programs, these units are assembled or compiled to produce object code. Listings are maintained and executable files (e.g. load tapes) are created based on configuration specifications. If the new units are user documentation, the master copies are updated. Then the system is installed on the target machine for designers to test.

This manufacturing cycle is best done on the development machine used by designers to create the software which will be executed on the target machine. These two machines may or may not share the same physical hardware. The facilities and operating systems used for the development of the software may be different from those used for the execution of the software as a product. The development machine should be optimized for increasing development productivity and software manufacturing. The target machine should be optimized for executing the product.

Notice that once a designer has turned over a new unit or update, Software Manufacturing owns it. From this point, the designer works from a copy and Software Manufacturing maintains the only recognized, official copy. Only Software Manufacturing may give systems to the test group for testing and later ship systems to sites. Programmers may not insert quick fixes in the official copy even under the threat of a schedule slip unless they are expedited through the Software Manufacturing mechanism. There are intermediate, measurable milestones, such as turnover to the Software Manufacturing group, integration and system test. Furthermore, by these functions being separated out, development groups are relieved of many of the mechanical aspects of the software developing process and management now has a way of managing by exception rather than monitoring each element of the job.

The tasks concerning quality control, common standards, and change control are related in that they require that an identification system exist. To go back to our first simple example in Figure 1, Figure 4 shows how that program can be identified as a piece of software. It now contains a name on the Pident line, where the Pident is the Product Identifier which names the software and identifies its version including a software change level. The change level in this case is A1. The date indicates when this software was last changed. Each line in this software unit is numbered sequentially and each time that line changes, the change level on the line is changed. All lines changed at that time are are identified as A1. The program is also titled and the programmer's name appears. The particular structure shown in Figure 4 is just one of many that can be used to format a computer program into a piece of software. Other formats are equally valid. The point is that this software now can be talked about as part of a larger entity and can be identified as to what its present configuration is, what its present state of change is, and who the author and the presently responsible programmer are. Figure 5 is an example of what might be standard requirements for program documentation and commenting.

Now that Software Manufacturing has a firm grip on its raw materials, let's look at what this group can do to help with testing. Testers can select from this data base according to their test schedules and be supplied with test data, system tapes, configurations, and listings. The software used to produce the test data may be under control of the software manufacturers and operated by them. The testers ability to be selective for system builds frees the design programmers from being tied to a test schedule. Since Software Manufacturing tracks and releases each unit and update, designers may turn over whatever they have available whenever they have it, regardless of test order, thereby avoiding that insidious disease, drawer rot. The Software Manufacturing group can maintain and run regression tests after each major milestone.

It might help to consider the organization of Software Manufacturing in the familiar schema of hardware manufacturing, where there are line functions for day to day production and staff functions for monitoring the general well-being of the product. The Software Manufacturing line functions can now be recognized as those tasks we had earlier identified as necessary in getting from the raw materials to the product.

After the software manufacturers are in place, they can take on added tasks of building tapes containing test data, running regression tests and, if management's tastes run to it, using specially designed software which checks compliance with coding conventions and program quality to process the source code.

The Software Manufacturing staff functions support both the line functions and the larger project management:

- produce reports for project management concerning the size of the job, outstanding problems, cleared problems, lines of code updated, etc.
- maintain hardware and software production facilities
- improve production techniques
- maintain project archives
- test conversions to various changes to the development machine (e.g., using new data base improvement systems, language processors, tapes).

To carry the analogy with hardware further, we can talk in terms of a software factory. The business of this factory which operates inside the larger project is to assemble piece parts into a system according to certain specifications. The software utilities needed in the factory depend on the degree of automation desired. The notion is the same whether this factory receives decks of cards as raw materials and ultimately ships

500 lbs. of materials for a 300,000 instruction real-time program (as was done in the '60s) or uses the sophisticated tools of a UNIX* system and Programmers Workbench (DOL 76A, RITA 74A).

Figure 6 shows an automated software factory. Programs are used to automate the update, assembly, trouble reporting and listing maintenance processes. The Software Manufacturing people maintain data bases on disk, and write utilities and job control code to permit the combination of steps in the manual process. Designers do not submit card decks, but rather inform the Software Manufacturing people of files in their private libraries which are ready to be transferred to Software Manufacturing's disk library.

Implied in Figure 6 is an approval boundary between designer and software manufacturer. Approval levels usually change during the project life, increasing in order to put more impedance in the way of changes and thereby stabilize the product. This is the prerogative of the project management, but the Software Manufacturing factory provides a natural point to impose control and have the people necessary to process the inevitible paperwork. It also serves as a check point from which deviations may be reported to management.

Figure 7 shows an implementation of this factory on the UNIX/Programmers Workbench (ibid) and the application computer. In this case, assemblers and loaders (driven by Software Manufacturing tools on the UNIX computer) operate on the same computer as the application but under a different operating system. The Programmers Workbench concept and the software facilities mentioned in Figure 7 were presented by T. A. Dolotta and others at the Second International Conference on Software Engineering (ibid) in October 1976. Here the software factory is combined within the development machine so that the program designers move with virtually no new training from the tools they use for design and implementation to those used by Software Manufacturing. In fact, the software manufacturers provide a service to the program developers by operating and maintaining their development machine. This leads to the mutual respect crucial to the programmers' acceptance of someone between them and the customer.

We have listed among the staff functions in Software Manufacturing the item of testing conversions of computers, operating systems and language processors. The factory organization is uniquely suited for such work. Conversions impede project development when designers are diverted to insure upward compatibility. In the worst case, upward compatibility is not exhaustively checked and updates become inconsistent with what is in the field. The automated Software Manufacturing factory can reassemble each program and compare the object code with that produced in the previous environment. Then only the differences can be reported to the design organization for resolution. Major projects have been unwittingly sabotaged by uncontrolled upgrading because the motivation of a good designer is to design the product, not test the new development tools. Software Manufacturing is in a good position to schedule upgrades, assess the impact, and do conversion certification.

Obviously this concept of Software Manufacturing requires a certain minimum starting environment and a project of large enough scale to make it economically feasible. We'll discuss <u>how</u> to get started in the next section. A practical indication as to <u>whether</u> projects should embark on this scheme would be to ask these questions:

- Is the project or group of projects at least large enough to require the full attention of one second level project manager, i.e., approximately 20 designers and programmers?

- Will future enhancements be made to the software by building on the established, working base?

- Will the project be delivered to one or more sites distant from the development site, but be maintained from the development site?

- Will the customer's employees be primarily responsible for running the system?

We suggest that if the answer to these questions is yes, there is sufficient work to constitute a full-time job for at least 2 software manufacturers. We say "full-time job" because, as we have argued, it is not feasible to split one person's job between this function and design work or clerical work because a different orientation and unique skills are needed. Additionally, it is probably wise to have at least 2 people engaged in this activity to cover contingencies, since this function is in the critical path for system testing and releases.

If, despite affirmative answers to these questions and the experience of losing control of software development, it is decided not to make the investment in Software Manufacturing, it would be better to let the designers be free to devise their own ways of getting the software out the door. Ingenuity and pride in this case will probably produce better results than imposing controls and expecting first line managers to enforce them.

<u>How Could The Software Manufacturing Function Be Introduced Into An Existing Development Effort?</u>

Any change requires that the project management have delicate sensibilities regarding designers' pride of authorship and natural reluctance to let anyone else tamper with their creations. The project management might begin by asking the following questions. If these cannot all be answered affirmatively, it would be wise to establish this minimum before proceeding further.

- Is the design modularized?
- Is there an existing build mechanism?
- Does an identification scheme exist or could one be established for each product element?
- Is someone responsible for each product element?
- Is each change related to an update report (or any other name: Action Request, Maintenance Request, Design Request, Enhancement)?
- Does the change mechanism require agreement by the people responsible for each product element affected, i.e., all related modules, all affected user documentation?

The emphasis on a change mechanism apparent in these questions is vital in maintaining order when software leaves the designer's hands. In some cases, it is only by management fiat, demanding that all changes be related to an update can be squelched. We observe, however, that given adequate tools, with a good notation scheme, the programmer designer who has project insight will act responsibly to determine modules can be altered and whether a fix is logical and consistent. The notation scheme simply relieves the designer of the mechanical effort of change and prevents surprises when other designers submit their work.

The project management must now make Software Manufacturing an honest profession by committing a respectable amount of resources and attention to the activity. It

*UNIX is a trademark of Bell Laboratories.

would be well to ease the transition by capitalizing on the present innovations of the designers and use whatever they have developed to improve edits, builds, etc. Software Manufacturing must have priority in computer time or they will bottleneck testing. This requires an initially high tolerance on the project management's part for the designers' cries of pain.
Organizationally, the Software Manufacturing supervisor must be equal to the design supervisors. At the beginning, the people working with that supervisor could be design programmers who are rotated through the organization. One can progressively reduce the education levels required for Software Manufacturing as those functions are streamlined - eventually to high school level, except for the person responsible for the continued improvement of the techniques.

The beginning activities for Software Manufacturing would be to produce a current listing of today's system with all fixes built in. Software Manufacturing would follow a daily build sheet from the design or test groups who set priorities based on project knowledge. From this beginning, other functions can be gradually added as the Software Manufacturing group demonstrates its capability.

Advantages of Software Manufacturing

The concept of Software Manufacturing frees programmers from production problems and knowledge of production environments. Coordination and communication through the organization is required which mitigates, to some degree, the indispensable person syndrome. It frees design management from the problems associated with managing production people and the production process. Production managers with a good understanding of how software is produced, but not necessarily a detailed understanding of how software is designed, can be used very successfully in these areas.

Software Manufacturing provides job opportunities for less educated and skilled people than those who are in a design organization. Technician and production level people would be given job opportunities in the software industry, giving growth paths to people now called librarians, clerks, software secretaries, computer operators. As this function takes on its own identity, it provides an independent line organization. By using people with appropriate skills to do the manufacturing, it can be accomplished more efficiently, at less expense.

Software Manufacturing permits management by exception by having a manufacturing group keep track of the development process and identify where things are going astray. By enforcing project standards and insuring that standards are carried across projects, several projects can use one Software Manufacturing entity.

The use of Software Manufacturing permits the introduction of assembly line techniques to make the problems of producing systems more automated. Programmers who are needed to automate the manufacturing process are industrial programmers, similar to the industrial engineers whose function it is to improve the methodology of producing a hardware product.

Disadvantages of Software Manufacturing

Software Manufacturing requires the introduction of a new organizational structure, which is difficult in an existing development effort. The subtle points of control and separation of control from the programmer can be traumatic and lead to the demise of a project if not done carefully. Therefore, to introduce this into an ongoing project, the steps must be gradual.

There is initial investment in setting up a new group and in identifying and training people for this function.

However, these costs are usually incurred by a project anyway and become very expensive in the long run, even if these functions do not show in initial budget estimates.

Probably the most serious disadvantage is that there are few people skilled in this kind of work in the computer industry. Those people who are so skilled do not want to give up the glamorous, well-paid task of designing the application computer programs. Yet to do the complete job as stated in our first objective without having manufacturing people identified, highly skilled and expensive people find themselves doing production functions for which they are unsuited.

Conclusions

Software Manufacturing as described here provides an organizational approach which is generic to a software development effort. Its introduction leads to systematizing the production of software, making software development people more productive, and therefore to better managed software efforts. This approach differs significantly from a cottage industry approach which relies on designers to perform all functions equally well.

REFERENCES

1. Doc 76A Dolotta, T. A. and Mashey, J. A. "An Introduction to the Programmers Workbench", Proceedings of Second International Conference on Software Engineering.

2. RITA 74A Ritchie, D. M. and Thompson, K. The UNIX Time Sharing System Comm ACM 17, 7 (July 1974) 365-75.

```
*THE INTENT OF THIS PROGRAM IS SOLELY TO
*FIND THE MAXIMUM OF A SET NOT IDENTIFY ITS LARGEST MEMBER

DECLARE A,B,C, LARGE FLOAT
INPUT:  A,B,C
IF A > B THEN LARGE = A ELSE LARGE = B
IF LARGE  < C THEN LARGE = C
PERFORM OUTPUT (LARGE)
END

OUTPUT (X):  PRINT, X
"THIS IS THE LARGEST VALUE, OF THREE NUMBERS ANY OR ALL OF THE THREE MAY EQUAL IT."
```

FIGURE 1

DEVELOPMENT CYCLE WITH SOFTWARE MANUFACTURING

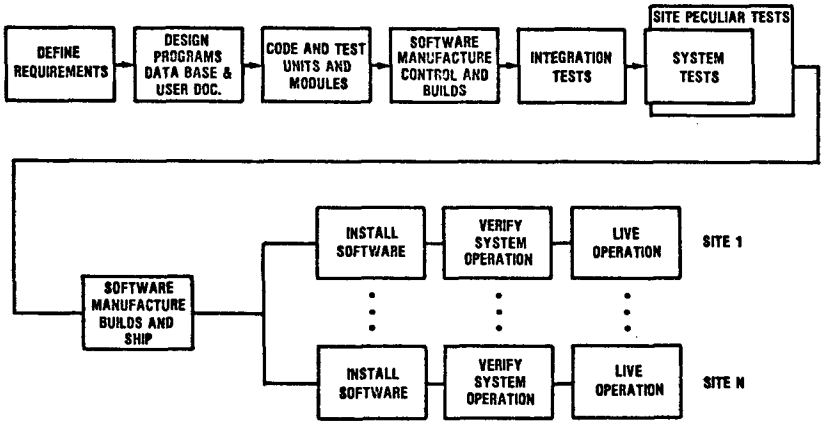

FIGURE 2

MANUFACTURING CYCLE

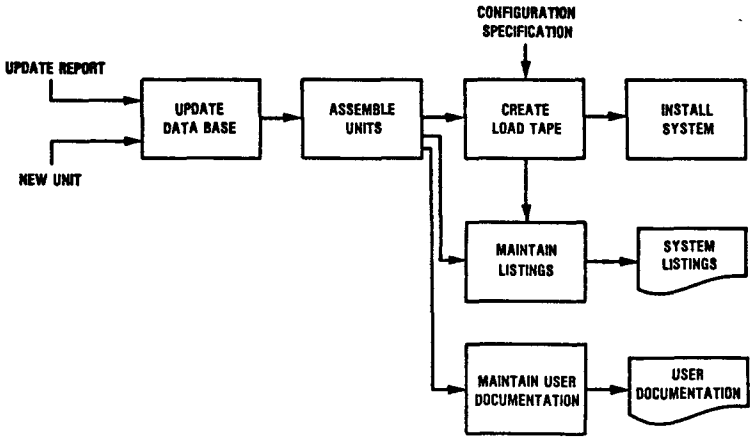

FIGURE 3

```
*PIDENT LARGEST.@VERSION01                    CHANGE.LEVEL A1          00100A1
                                                                       00200A0
*SARAH JONES          11 NOV 74                                        00300A0
*JANE SMITH            6 SEPT 76                                       00400A1
*TITLE - LARGEST.VALUE.IS.A.SET.OF.THREE                               00500A0
DECLARE A,B,C,LARGE FLOAT                                              00600A0
INPUT A,B,C                                                            00700A0
PERFORM CHECK (A,B,C)                                                  00701A1
IF A > B THEN LARGE = A ELSE LARGE = B                                 00800A0
IF LARGE < C THEN LARGE = C                                            01000A0
PERFORM OUTPUT (LARGE)                                                 01100A0
END                                                                    01200A0
OUTPUT (X):  PRINT, X ("LARGEST OF A SET OF THREE")                    01201A1
END                                                                    01300A0
CHECK (A,B,C):  IF (A,B OR C = NUMERIC) END ELSE PRINT "ERROR",        01400A1
                "INPUT A = "A, "B =, "B, "C = "                        01500A1
END                                                                    01600A1
```

FIGURE 4

FIGURE 5

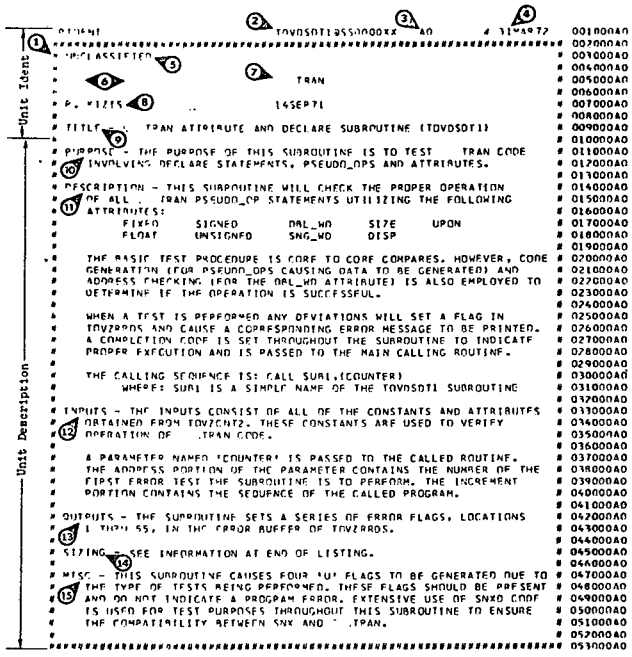

FIGURE 5 (continued)

DESCRIPTION OF CONTENTS

The following describe what is to be included under each of the numbered items.

UNIT IDENTIFICATION SECTION

1. Comment Indicator - a comment indicator consistent with the language must be used.

2. PIDENT - the identifier used as the name of the Program Unit. (Note that the word "PIDENT" is required at the beginning of the card as shown.)

3. Change Level - the designation used to identify a particular assembly of a program.

4. Date - the original assembly or compilation change level date of the Program Unit. The date is recorded in the following format: DayMonthYear (06Sep71)

5. Classification - the degree of privacy desired enter - Unclassified, Company Proprietary, Trade Secret.

6. DPS Number - the Generation Breakdown Number of the Program Unit

7. Language - the compiler or assembler which is to be used for the Program Unit, e.g., TRAN.

8. Person Responsible - the name of the person currently responsible for the Program Unit and the date responsibility was acquired. The card indicating the change of responsibility shall be included with the next scheduled revision. (When a transfer of responsibility takes place, each card indicating the previous person responsible will be maintained in the deck in reverse chronological order.)

9. Title - the Program Unit title that gives some indication as to what function the program performs. This will be treated as the official name of the program.

UNIT DESCRIPTION SECTION

10. Purpose - a brief statement describing the function of the Program Unit.

11. Description - a general description of the Program Unit including the calling sequence, information concerning each entry point, initialization requirements, and in cases where the program is complicated, a brief description of its organization by subfunctions.

12. Inputs - all information supplied to, and used by, the Program Unit via registers, data sets, and the stack. All variables read from a data set must be listed here unless the entire data set is read, in which case it should be so stated.

13. Outputs - all information that this Program Unit passes to another program via registers, data sets, or the stack. All variables written in a data set must be listed here unless the entire data set is written, in which case it must be so stated.

14. Sizing - currently, the size of the program in appropriate units is automatically printed at the end of the listing, i.e., 64-bit words for TRAN and bytes for PL/1, FORTRAN, and BAL.

NOTE

A future revision to the language processor for TRAN will automatically print size information within the Preface.

15. Miscellaneous - any information that either the group's Supervisor or the programmer feels would aid in using, understanding, or debugging the Program Unit.

THE SOFTWARE FACTORY

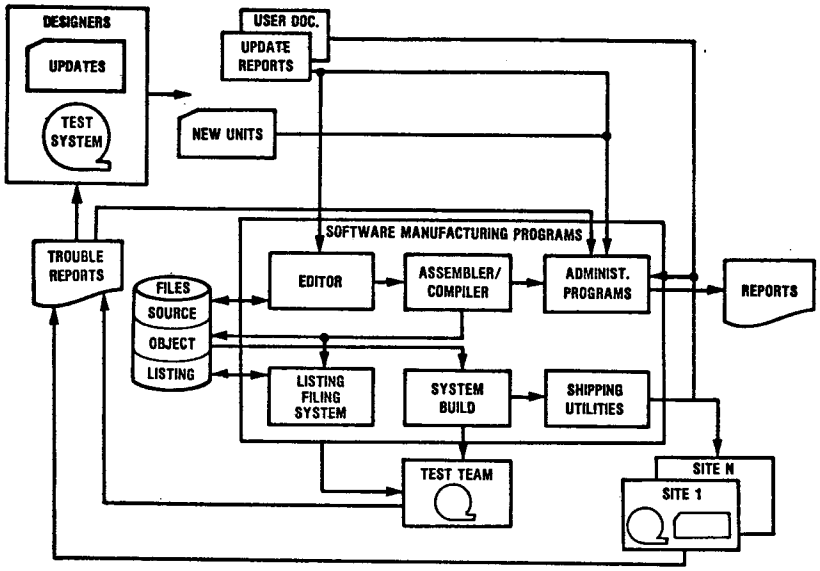

FIGURE 6

THE SOFTWARE FACTORY IMPLEMENTED WITH UNIX/PROGRAMMERS WORKBENCH

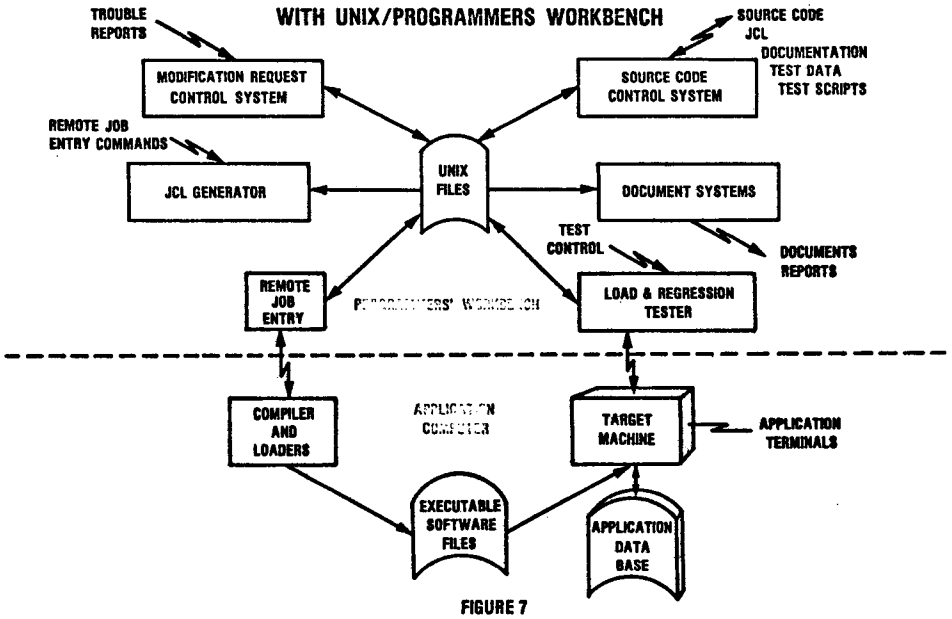

FIGURE 7