An HP 3000 Approach to IBM's LIBRARIAN Techniques

Betsy Leight
OPERATIONS CONTROL SYSTEMS
560 San Antonio Road
Palo Alto, CA

As HP data center become more sophisticated, users are attempting to introduce more standards and controls into their environments. During the early stages of growth, the need for controls resulted in automated batch processing, access restrictions, menu drivers, and non-user scheduling. In today's environment more sophisticated concepts such as file management and accountability are beginning to come under scrutiny. File management concepts are not new. In fact, IBM mainframe users have been controlling their development files for years with the aid of PANVALET and ADR/LIBRARIAN. These products separate test and production files, control source code libraries, archive modules, and perform audit functions. Many HP users now recognize the benefit to be derived from these techniques.

In this paper I will discuss the problems inherent in current control techniques and describe possible solution pathways, however, it is essential to establish unequivocally that the current approach to development tracking is inadequate. Whether it is recognized as such or not, one of the major functions of any data processing department is the creation and maintenance of software. This is not a process confined to development houses alone. Every day, any MIS department could receive requests to modify software and data. The efficiency and cost of development are instrumental to the corporation's success because computer applications have become an integral factor in the competitive struggle for market position. The software alone can represent corporate assets valued at hundreds of thousands or even millions of dollars.

Surprisingly, there are many environments that have no source access or change restrictions. Programmers can access production source directly. This approach should never be allowed, for two reasons: First, the original source code can be destroyed. If a production error results from programmatic changes, the original version is not easily restored. Therefore, production can be delayed for several hours, or in the worst case, for days. Second, there is no audit track. Programmers are not restricted from making unauthorized modifications directly to production files. Doesn't this possibility worry you?

Even in cases where some restrictions exist, other problems can arise.

Current development procedures often result in a discrepancy between source and object code. In other words, the master source file does not recompile into the production object file. Since it is extremely difficult to recreate source code from a load module, there is no way to ascertain that all the object code features exist in the source. Should the source require a change, there is no guarantee the resultant object will have the same functionality as the original load module. Production can and does often fail as a result.

Another common scenario involves multiple programmers who make change simultaneously to the same source module. In this case, the last programmer to update production wins because previous fixes are obliterated. The net result is wasted effort and invalid object code.

Although these and other problems are obvious to many, a dichotomy begins to appear when it is discovered that many HP 300 centers continue to operate in a reactive mode without introducing standards. Direct user support is considered the primary function of development and operations. This translates to a daily goal centered on processing user-requested jobs, completing batch production, and distributing reports. Although these functions are routine tasks of operations, one should not overlook the basic fact that the efficiency and accuracy of operations' output rely upon the cohesiveness of the software components. Just as a solid house cannot be built upon shaky foundations, reliable computer processing cannot be achieved unless the associated code shows internal integrity.

The component integrity problem is one HP 3000 centers are not managing effectively. Instead, the issue is currently relegated to the "wish list" and usually escapes notice. This policy is not a solution. Although operations can proceed error-free for months, inevitably a simple oversight snowballs into a catastrophe.

The important point is that software development and file maintenance procedures directly affect operations. Should programs fail, run out of order, or produce invalid data, the operations department will be required to rerun procedures. To avoid erratic production problems, file management issues need to graduate from the "wish list" to the "current projects list"

It would be inaccurate to give the impression that HP users are not addressing this problem at all. Some are, but the degree and depth of resolution vary widely. Let's examine some checkpoints along this range.

Various Approaches

The simplest procedure is no procedure. In other words, programmers merely log directly into the production account, modify the code, test it, and recompile. All three steps occur within the production location. As mentioned earlier, this method has no safeguards and is extremely risky.

A slightly more sophisticated approach requires programmers to FCOPY or CHECKOUT source code from the production account and move it into a development location. Unless stringent controls exist, there is no guarantee that only one individual has checked out a particular module. Furthermore, programmers are not restricted from accessing production accounts, nor is there any way to audit their access.

The production-to-development strategy just described can be envisioned at three levels. At the lowest level, the development area may be nothing more than an amalgamation of programmer groups. In this case, each programmer copies, develops, and tests in his own group. There is no standardized testing environment. A second level maintains a development account that duplicates the production account. Thus, each programmer can be confident that alpha testing is occurring in an environment that closely resembles production with accurate, up-to-date object and load modules.

Ideally, production and development accounts should exist on separate CPUs to eliminate completely the possibility of direct access to master files. Of course, this approach is not always possible and separate accounts on one CPU will produce adequate results.

Once the development effort has been completed, what happens to the code? My experience suggests the same developer simply overlays the original production files with the enhanced code. Such alpha testing does not represent adequate control because the programmers who test their own work can easily overlook bugs. Besides, they tend to test what works rather than attempting tests to "break the code". For this reason, it is strongly suggested that a Quality Assurance (QA) process be initiated.

There are several ways to initiate such a procedure, depending upon company size and resources. Smaller companies may have alternate programmers QA test their colleagues' development efforts. The retests are usually performed in the development account. Larger organizations may hire an individual or staff whose sole function is QA testing. When the process develops to this extent, there is generally a separate QA test account that reflects both the production and development environments.

Should a QA process exist, it is vital that the developer relinquish all claim to the code when it moves to the QA phase. Only one copy of the developing code should reside in either the development or QA area. If both accounts contain separate copies, undocumented changes to the development files may not be incorporated in the QA version. Thus, the final production version would not include all code changes. This safeguard is commonly overlooked. Similarly, if the QA analyst locates a discrepancy in the modules tested, the code should be returned to the original developer for revision. It is now QA's turn to relinquish all claim to the code until it is returned by the programmer.

At the conclusion of QA analysis, another step can exist. A higher level manager should perform a final approval on the development effort to ensure all checkpoints and tests have been satisfactorily completed.

Followed final approval, the enhanced code is ready to be moved into the production location. An FCOPY or move will overlay the original modules. The destruction of the earlier version could be detrimental if the revised code contained errors and no backup copy of the original files existed. Therefore, the original modules must be stored to tape prior to enhancement installation.

The update step can be both time-consuming and error prone, especially when large numbers of files are involved. To make matters worse, a compile step and JCL update must also be coordinated. Standards cannot be eliminated at this final stage. If they are, production may be under old JCL or inaccurate object code resulting in production that aborts in the middle of the night. Sound familiar?

When they exist, the aforementioned procedures are usually tracked on paper with a form. An originating service request often moves with the code from production to the programmer to the QA analyst. Each step along the process is signed off on the form. Such a tracking method is inadequate for several reasons. Most simply, the paper can be lost or misplaced, or any member of the chain can fail to record his involvement. Most importantly, there is no assurance that the form really reflects what has occurred.

Individuals have been known to misrepresent information for a variety of reasons -- often in the name of speed. A manual paper tracking method can be synonymous with no tracking method. An appropriate, complete, and accurate tracking procedure should be a primary concern to the development and audit staff.

Obviously, development solutions are very flexible. They can evolve through a number of steps and can encompass varying levels at each step.

An idealized strategy

It is possible at this point to extract an idealized development strategy based on the previous discussion. Needless to say each environment will require additions to or deletions from this ideal. However, the following does describe a general goal based on my own experience.

In this idealized scenario, three accounts exist: a production or master, a development, and a test account. Each account structure is a carbon copy of the others to the extent that files moved from location to location retain their original jobname and group designators. Only the account names change. In this way, it is easy to visualize the link between a developing program and its originating master.

When files move between the production location and the development area, a copy should be made. The original source should never be destroyed. However, movement between development and test should result in only one copy at either location.

Once QA has approved all changes, a project leader or manager should verify that adequate and accurate test procedures have been followed. Only at this point should code be moved into the production library. Such updates could occur once a day if desired and should be performed by operations or a production librarian. The latter is my recommendation as it restricts the responsibility, control, and audit functions to one individual.

Prior to update, the original production should be verified and stored. Without this step it is much more difficult to return to a prior version in the case of error.

Software tools that perform file tracking and auditing procedures automatically without information loss are available. The tools also force participants to conform to structured rules to ensure steps are performed in sequence along the development pathway.

In order to implement a structured development strategy with some components of the idealized route in your environment, it is vital to define goals. Possible goals include but are not limited to:

1. Establishing controlled access of production modules.

2. Creating a set of rules to minimize file transfer and maximize efficiency.

3. Ensuring a link between compatible object and source code.

4. Verifying all associated files, such as JCL and databases, are saved and moved to production concurrent with source and object updates.

5. Developing a methodology to track file movements accurately.

Development strategies, an idealized solution, and the goals to consider in achieving the ultimate solution have been described. To configure your site along the ideal path, four steps are necessary. First, identify and flowchart the specific attributes of your best solution to software development based on your needs. Second, assess the components of the development strategy that are currently employed. Third, compare the current structure to your idealized goal and prioritize change requirements. Fourth, develop a project plan from the priority list and implement the necessary changes.

Unfortunately, it is not possible to provide flowcharting assistance in this article. Each development effort is unique. However, the scenarios previously described should provide hints and suggestions for the first step.

Several areas of concern can be identified during the evaluation process of current strategies. Questions that should be asked during this analysis are included here.

After these data are collated, it is possible to perform a comparison between current methodologies and the site specific optimal development path. This third or comparative step is, once again, subjective. Each individual must determine for himself where the current procedure diverges from the idealized goals.

Prioritization of differences is dependent on site-defined needs. For example, the auditors may be clamoring for file movement control. Thus the implementation of tracking procedures would be the primary concern. On the other hand, QA testing can be the vital link. In either case, auditors can prove to be a wonderful resource in this evaluation process. The comparison step is often the easiest in the four-step strategy. It draws results from the analysis required to accomplish the previous steps.

analysis required to accomplish the previous steps.

Following prioritization, implement a project plan to attain the stated goals. Generalized solutions should have evolved through the process of current assessment, goal derivation, and priority setting.

To summarize, the need to control and track the software development cycle in becoming increasingly apparent. Without standardized controls it is virtually impossible to establish the validity of software modifications. This is most important in larger environments, especially those that manage enhancements for remote locations.

Therefore,it is important to move toward an idealized software development process. This goal can be accomplished by comparing the optimal solution to present controls and implementing plans to minimize strategy gaps. Manual tracking procedures can be used for this purpose, if necessary. Fortunately, the industry also offers software tools for an automatic solution.

Evaluating Software Development

1. FILE LOCATION: Where do the production files reside? Are all files contained in one account? Several accounts?
2. DEVELOPMENT ACCOUNTS: How are the development areas configured? Does each programmer maintain a unique development group or does all development occur within the same group?
3. ACCESS RULES: What rules are implemented to control file movement between production, development, and testing? Who has access to these locations? When does access occur? Are there any preconditions, such as management approval?
4. ACCOUNT STRUCTURE: What is the structure of the production, development, and test accounts? Are they duplicates of one another?
5. VISIBILITY: How much visibility of file movement exists? Can only one programmer gain access to a file at any time? Can all changes made by each programmer be verified in the content of the final code? Does a validation check for compatibility of source and object occur?
6. QA ANALYSIS: Is there an established need for separate Quality Assurance testing? Does the current development effort include QA testing? Are there plans to move in that direction? Where will QA testing occur? Does the account structure duplicate production and/or development accounts?
7. FINAL APPROVAL: Does tested code pass through a final checkpoint prior to re-entrance into the production account? Who is responsible for this final check?
8. UPDATE STEP: Who is responsible for moving tested code into the production account? At what time(s) does this occur? How much error exists within the current strategy? What can be done to reduce inaccuracies at this step?
9. VERSION CONTROL: Are original production files by newly modified code? Where do copies of old versions reside? Are all versions verified and tracked? What type of recovery procedure is available if new code fails? How does this occur? How complex is the recovery?
10. AUDIT TRACKING: How are file movements tracked? Is there visibility of when, why, how, and by whom files are moved? Can inadvertent purges be identified?