LA-UR -86-964

CONF-8604151--1
Received by OSTI

APR 07 1986

LA-UR--86-964

DE86 008743

TITLE: A MODULAR, AUTOMATED SOFTWARE TESTING ENVIRONMENT

AUTHOR(S): G. Cort and R. O. Nelson, P-9

MASTER

SUBMITTED TO: Softool Users' Group Meeting
14-15 April 1986
Culver City, CA

DISCLAIMER

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

A MODULAR, AUTOMATED SOFTWARE TESTING ENVIRONMENT

G. CORT AND R. O. NELSON
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS, NEW MEXICO 87545

Background

The testing environment has evolved from the existing LANSCE software development methodology in response to some deficiencies identified therein. Principally, it had become evident that the weakest link in the development cycle occurred in configuring a certified software product for operation. This problem derived in large measure from the structure of the methodology.

The Los Alamos Neutron Scattering Center (LANSCE) approach to software development and software configuration management are described in detail elsewhere.[1,2] However, in order to provide the necessary context for understanding the testing environment, the salient features of the LANSCE hybrid approach are described in the following paragraphs.

The LANSCE system provides an evolutionary software development methodology which incorporates a comprehensive, automated configuration management system based upon Softool's Change and Configuration Control (CCC) environment. Fundamental to this approach is the exclusion of all development activities from the CCC data base. The contents of the CCC data base are restricted to certified baselines (i.e. software that has passed the appropriate reviews). Development activities (including software testing) are performed in the host operating system accounts of the various programmers. A very powerful set of software tools (CCC macros and VMS command files) support this methodology, thereby allowing the configuration manager to transfer files between the development and configuration management environments.

The advantages of this strategy are numerous.[3] Programmers are given maximum flexibility to develop and implement their designs, yet certified software is captured and controlled effectively. In addition, data base size is minimized and the computing system resources diverted to support the CCC environment are drastically reduced.

The software tools provided to support file transfers between the development and configuration management environments have also proved highly effective. The two major functions of these tools are to release selected modules of an application from the configuration management environment to a particular developer (as the first step of a maintenance or enhancement activity), and to move modules from the development environment to the secure configuration management environment after the final delivery acceptance review has been satisfied.

Software releases pose few problems for the automated system. In response to an engineering change request or trouble report a programmer submits a request for specific modules of the affected application. Upon approval of the request, the configuration manager invokes a CCC macro which constructs the appropriate export lists and transfers the required modules to the developer's account. These modules generally include documentation and testing software as well as the source code to be modified. Even if the request is not complete, no serious harm is done. A subsequent request can be submitted for additional modules which can be similarly withdrawn from the CCC data base.

In the case of submissions of certified software to the configuration management environment, however, a far more serious situation can arise. Again, the process is initiated by the developer with the submission (by electronic mail) of a transfer request which specifies the source code, documentation and testing modules to be moved into the configuration management environment. This request is made after the final review of software, documentation and test report has certified the functionality of

the software being submitted. The configuration manager is then responsible for transferring and cataloging the software in the configuration management environment as well as for building the associated application and for updating system data bases to make the applications generally available to the user community. Each of these tasks is performed automatically by the appropriate software tools.

Two problems can arise at various stages of the procedure. The first (and least serious) is associated with identifying inconsistencies in the delivered software which prevent the application from being compiled or linked in the more general system environment. Such local references are invariably discovered and reported by the automated procedures which build the applications, and the submitter can be prevailed upon to effect remedial action. Local references are generally relatively trivial in nature (incorrect logical name usage, improper defaulting in file references, etc). They are easily identified and present no challenge to repair. However, they can prove an annoying disruption to the submission process,and over the long term, can impose a significant overhead in terms of wasted effort and bad feelings between the configuration management and development communities.

The second problem which can arise during the submission cycle is potentially much more serious, owing to the fact that it is far more difficult to detect, and because components of the baseline can be lost as a result. This problem derives from an oversight on the developer's part when specifying the modules to be transferred to the configuration management environment. Although a missing source code module is readily identified by the automated tools, more subtle omissions (one of many files of input test data, for example) cannot be detected automatically. The only sure defense against such an occurrence is for the configuration manager to execute the testing software after the modules have been transferred to the configuration management environment. Unfortunately, a special hardware configuration is often required for operation of this

software. As this hardware is generally not available on the configuration management computer, this approach cannot be implemented.

The most serious aspect of this problem derives from the fact that upon being notified of the completion of the transfer operation, the developer generally deletes the associated software from the local operating system environment. Files that were omitted from the transfer request are now lost; their absence may not be discovered until the next maintenance activity (which may be months or years in the future).

The LANSCE modular testing environment is designed to address both of the problems associated with post-review software submissions. This is accomplished by providing an isolated environment, segregated from the developer's local work space, in which all testing is performed. In this manner, local references can be identified and implemented by the developer at an early stage of the testing activity. In addition, the independent environment removes the human element from the specification of those modules which are ultimately transmitted to the configuration management environment. In this manner, the submission process is completely automated, thereby eliminating the possibility of omissions.

Implementation

The testing environment is established in an independent host operating system account on a computer with the appropriate hardware configuration to support the software to be tested. It is supported by a comprehensive set of automated, software tools which significantly enhance the user interface to the environment. It is utilized by the developer to perform final acceptance testing of a software application prior to submission to the final delivery acceptance review, and as the source of all modules to be transferred into the secure configuration management environment.

As specified by application-dependent characteristics, the developer invokes one of several software tools to configure the environment for testing a particular software product. As part of the process, the developer specifies the name of the source file list[3] (SFL) for the application source files as well as file names for users' documentation and testing software.

A subdirectory is automatically created within the test environment and the specified software is identified and automatically transferred to the subdirectory. Special symbols and logical names are created to support the testing effort and all software (including test drivers) is automatically compiled and linked. This process identifies any local references which exist in ary source code module. A brief description of each local reference is formatted into a message which is sent to the corresponding developer by electronic mail. This message generally provides sufficient information to identify and eliminate most such references. It references a history file which contains detailed information about all phases of the software transfer and build process and can be used in the event that the information provided in the brief message is insufficient to diagnose a problem.

The software tool which performs the software transfer and rebuild is characterized by a simple, high-level user interface. The interface is designed to minimize the amount of interaction required by the user to configure the environment for testing a particular application.

The interface prompts for all information. The need for the user to specify large numbers of source files is eliminated by driving the system from source file lists. In this way, the user need specify only the name of the application's source file list in order to transfer and rebuild the entire application. This streamlines the interface and virtually eliminates problems associated with incorrectly typed file names and inadvertent omissions.

Although the vast majority of files to be transferred to the test environment are specified through a single reference to the SFL, other files exist which must be explicitly identified by the developer. Among these are documentation files, command files and testing data bases. In order to simplify the specification of these files, the interface supports a "wildcard" mechanism which is identical to the corresponding feature of the host VMS operating system. Thus a single wildcard specification can result in the transfer of an entire class of files to the test environment.

The test environment also provides a facility which automatically rebuilds and executes the test suite. In so doing, a formal test report is automatically created and cataloged within the appropriate partition of the environment.

Interface with the CCC System

The LANSCE configuration management plan specifies the use of the CCC configuration management environment to control and track all software baselines. This is accomplished by employing a set of software tools to transfer the components of a certified baseline into the appropriate partition of the CCC data base.

Previously, the files to be transferred were specified by the developer on a special form submitted via electronic mail. Because the specification formalism incorporated SFL's for referencing source code modules, these components could be identified and transferred very reliably. However, because testing and documentation files cannot be referenced through source file lists, there existed the possibility that some (or all) of these components could be inadvertently omitted and thereby vanish from the hierarchy of controlled software.

The use of the testing environment eliminates the potential for losing files in this manner. In addition, it permits the transfer of baseline components to the CCC data base to be performed completely without human intervention.

At the conclusion of testing and the final delivery acceptance review, the testing environment contains a partition in which all of the components of the corresponding software application reside. These components are presorted into subpartitions which correspond to the low-level organization of the CCC data base. As such, subpartitions exist for application source code, the test suite (including the test plan and test report) and users' documentation. Because both the application and test suite have been built and executed in this environment, the software and supporting data in these subpartitions are guaranteed to be complete.

Software tools (in the form of CCC macros) are provided to admit all software for a specified application into the CCC data base. To initiate this process, only the application name need be specified -- no names of individual files are required. These macros automatically create the appropriate data structures within the CCC data base to receive the baseline components. The HOST facility is then employed to scan the appropriate partitions of the test environment, compile a set of import lists and move the corresponding software into the CCC data base. A separate set of HOST macros is employed to rebuild the application and to integrate it into the LANSCE software environment. This integration includes updates to object libraries, command tables and the LANSCE data base of executable images. A final set of macros notifies the developer of the admission of the baseline to the controlled environment and deletes the corresponding modules and partitions from the testing environment.

Conclusions

Use of the modular testing environment described in the preceding sections has several significant advantages. The first such advantage derives from the simplification of all aspects of the software testing and submission process.

From the developer's perspective, the approach provides an automated facility for configuring the software for final testing and for eliminating local references from the application. The manual methods previously employed to accomplish these tasks were unreliable, tedious and error prone. In addition, provision of these automated facilities permits the transfer of the most difficult aspect of final software integration (namely, the identification and resolution of local references) to the individual most familiar with the application: the developer. As a result, local references can be more efficiently resolved than when their discovery is postponed and relegated to configuration management personnel. In addition, because all local references are eliminated before the software is submitted to the configuration management environment, integration of the application can be performed completely by automated tools, thereby further enhancing the efficiency and reliability of the operation.

The introduction of large scale automation into the software submission process also provides numerous benefits. Principal among these is the elimination of potential problems relating to completeness of the submission and resulting in compromises of the integrity of the configuration management environment. In addition, because the automated operation no longer requires human interaction, it can be scheduled to execute in a background mode during periods of low utilization of the host computer resource. In this manner, the high demands for system resources made by the CCC system can be scheduled for periods when they will have the least impact on other system users.

Finally, it should be noted that because of its modular implementation, the testing environment is suitable for use by projects of

widely varying size and organization. In the limit of small projects such as the LANSCE effort, this approach is very effective for providing individual developers with a uniform, independent environment in which to integrate and test their software applications.

It is our belief, however, that an approach such as this is potentially even more beneficial when applied to large software development projects with independent testing organizations. In addition to providing automated support for both unit testing and integration testing activities, the environment provides the means to test evolving software configurations (and to freeze intermediate stages of development) without intruding upon the activities of the development community. Regardless of project size, this approach to software testing and integration relieves people from many tedious, time-consuming and error prone tasks, with the ultimate result of increasing efficiency, productivity and morale.

References

1. G. Cort and D. M. Barrus, "Configuration Management for Mission-Critical Software: The Los Alamos Solution." Proceedings of the Softool Users' Group Meeting, September, 1984.

2. G. Cort, J. A. Goldstone, R. O. Nelson, R. V. Poore, L. Miller and D. M. Barrus, "A Development Methodology for Scientific Software." IEEE Transactions on Nuclear Science, NS-32, 4, 1985.

3. G. Cort, "The Los Alamos Hybrid Environment: An Integrated Development/Configuration Management System," in Conference on Software Tools, John Manning, ed., IEEE Computer Society Press, 1985.