CONFIGURATION MANAGEMENT IN A SOFTWARE DEVELOPMENT ENVIRONMENT

T. Michael Moriarity
AAI Corporation
Cockeysville, Maryland

ABSTRACT

Software configuration management deals with the identification, change control, and status of software items. Properly applied configuration management procedures can improve visibility, efficiency, and integrity of the development process. In software intensive development projects, configuration management principles in general and change control practices in particular are sometimes perceived as obstructions to the development process. Therefore, these practices are often implemented late in the development life cycle. The advantages of configuration management are thus lost during the early critical phases of a program.

Part of the reason that configuration management procedures are perceived as an impediment to the development process is that traditional configuration management practices do not provide timely benefits commensurate with their maintenance effort. Traditional practices were developed on large production programs in which the emphasis was on the control and accounting of a relatively small number of changes to a relatively stable product. The software development environment, on the other hand, is characterized by the integration of a large number of changes into a transient product.

A configuration management system designed to be more compatible with the software development process and to take advantages of software development resources can provide extensive benefits to the development team. These benefits may encourage the installation of configuration management procedures early in the software development phase, thereby providing the advantages of configuration management throughout the development life cycle.

This paper draws on the experience of several large simulator software development programs to identify design strategies for a configuration management system that provides substantial benefits during the development phase and meets the requirement of traditional configuration management in the latter phases of the development life cycle.

Introduction

In simplest terms, configuration management is the identification of items to be managed, the control of changes to those items, and the reporting on the status of installation of authorized changes. Software items that are configuration-managed usually include design specifications, computer readable files (routines, data, job control streams, etc), and user documents. Identification consists of determining which items are to be managed, assigning a unique identifier, and describing the item in terms of their functional characteristics. Change control traditionally implies establishing procedures to review and authorizing changes to managed items before the change is actually implemented. Reporting requires the setting up of a recordkeeping system that records the authorized changes and reports on the status of the incorporation of the changes into the end product. Control and reporting imply the establishment of baselines or accepted starting points against which changes are made. Typically, several baselines are established throughout the life of a project.

The purpose of configuration management is to help insure the integrity of the system under development throughout its life cycle, i.e., to insure that all items that make up a deliverable product have been identified, that the product was built as specified, and that another unit can be made from current documentation. In typical development programs, configuration management starts with the maintenance of the procuring specification, follows with the development and maintenance of intermediate design specifications, and continues with the identification and control of the items to be developed. As the program matures, test plans and procedures, user documentation, and installation documents are also managed.

Configuration management began in the early 1800's with the advent of the industrial age when manufactured parts were mass produced rather than handmade and individually fitted together. With the mass production of interchangeable parts, a discipline had to be introduced into the manufacturing process to identify each part; to control changes to form, fit, or function of the part; and to keep track of the changes as they were accepted and implemented. The management of configuration of parts was refined by Henry Ford in his automobile production lines and was further extended by industry during and following World War II. As the complexity and technology of manufacturing systems increased, the management of change became more important.

By the early 1960's, the Department of Defense had issued directives and instructions to implement configuration management procedures in a wide range of military procurements. These early directives were based on the experience of successful programs used in industry and, as expected, were generally concerned with hardware. It was not until the 1970's that a real concern for software configuration became evident. This concern coincided with a larger effort to apply proven engineering principles to the development of software. The effort was fueled by the realization that sound engineering design principles, one of which was configuration management, could be applied to software as they had been to hardware. Configuration management procedures, however, were heavily influenced by the mechanical production environment from which they came; they neither took full advantage of the resources available nor met the needs of the software development environment.

Traditional Versus Software Configuration Management Systems

There are several differences between the traditional mechanical production environment and the software development environment; these differences have an impact on configuration management. A primary difference between hardware production and software development is the differentiation of

labor. In the hardware environment, one group designs an article, a second group actually produces the item, and a third group uses the part in an assembly. The differentiation of labor requires detailed specifications or directions to be written before the next group can perform their task. In software development, a single person will often perform all three tasks. Specifications in this case are used more by the buyer or user to describe desired characteristics of an item than by the development group to define a production step.

Another difference is that in the software development effort the configuration-managed item is in a form that can be managed directly. A computer file can be managed similar to the way a drawing or specification in a hardware environment is managed. Detailed specifications or drawings identifying and describing the file need not be created because the file can be self identifying, containing within itself its own identification and description.

A third difference between the hardware and software environment is that the configuration management tools and processes used in the mechanical environment are separate and removed from the tools of production. The identification, control, and accounting of the specification, which, in turn, controls the fabrication of a part must be processed manually on a clerical system separate from the mills and lathes on which the part is machined. In the software development environment, the computer, which is the major tool for producing software, often has the resources to maintain configuration control.

In summary, in the software development environment, more emphasis can be made on identifying and controlling the managed item directly as it is being developed rather than on the specification of how the item is to be built. Traditional configuration management controls how an item will be built, whereas software development practice monitors how an item is being built.

A good configuration management program can improve visibility, encourage coordination, and insure integrity of a system. Visibility is particularly important in software projects where the product is largely intangible. Visibility is improved by establishing specification baselines and clearly identifying software items needed to fulfill the specification requirements. After software items are developed, configuration methods can provide a mechanism for coordinating the changes that are a result of integration and testing activities. Adherence to basic configuration management principles will help insure that the system was built as designed and that the system can be replicated in the future.

The benefits of configuration management make it advantageous to implement a configuration management program early in the development cycle. Increased visibility, coordination, and design integrity will greatly increase the efficiency of the development process, however, a configuration management program is often not implemented until late in the development cycle when the benefits may not be effective. The reason that this occurs is that the costs of a traditional configuration management program may outweigh its benefits, particularly in the beginning of the development cycle. The usual configuration management program does not produce the detailed information needed by the development team or the information that is provided is not available soon enough to be useful. Typical of the information needed during development is the answer to the question, "What changes were made to routine 'XYZ' last night?" Unless the routine had been processed through a configuration management system, the answer would not be known. If the routine had been processed through traditional configuration management procedures, the answer may not be available for several days.

The cost in terms of extra work load on the development team and delay in schedule is usually too high to allow incorporation of a traditional configuration management program until late in the development cycle when the volume of changes is reduced. If an analyst has to fill out paper work and wait for approval for each change, the development process, which is highly iterative, is greatly impeded.

The high cost, low benefit result is largely due to the fact that traditional configuration management procedures are often lockstep, manual procedures derived from earlier production systems. A configuration management program more in tune with the needs of the software development environment will result in high benefit at a much lower cost.

A Software Configuration Management System

The following is a brief description of the salient features of a configuration management system that has been used on several large simulator programs. The key features of the system are: it resides in the computer in which software development occurs; it controls the managed item or file directly; it is largely transparent to the user; and it is flexible in the control exercised throughout the development cycle.

The system consists of a central file called a Configuration Control File (CCF) and several software processors that interact with the CCF and the files to be managed. The CCF lists and describes each software item to be controlled on the project. These items include source files for programs and routines, data files, job control streams, etc. In one large project, all design specifications and user documentation were also maintained as files in the computer. A record in the CCF is generated for each item to be controlled. An individual record contains entries such as file name, identification number, responsible analyst, software family tree descriptor, brief description, applicable language, and library information.

A collection of private and system directories is set up with various access priviledges. Individual analysts work in private directories to create files and to move them between directories, using various interactive configuration management processors.

The first step in the configuration management process is the identification of files to be created. This occurs as soon as a file is defined in the design process. After it is defined, the analyst provides the name, place in the software family tree, and a brief description. The program librarian or other project personnel adds additional information such as identification number, library information, and documentation requirements. The program librarian checks to insure that naming conventions are correct and that all required information is provided. The program librarian then enters the data in the CCF. After the file is created, data can be selected, sorted, and printed in a number of reports. A typical report is a software family tree.

Additional software programs can be created as development tools. One that has been found useful is a preamble generator. This processor accesses information from the CCF to generate a skeleton preamble or header for a routine. The header includes the routine name, identification number, brief description, responsible analyst, language, etc. The processor can also generate major sections of the routine by generating documentation sections (purpose, history, description, etc), data sections, code sections, and a return statement.

The adding of routines to the CCF is loosely controlled in the early stages of the project. Analysts are strongly urged to submit CCF entries on transmittal forms as early as possible even though they may be deleted or changed as the design is refined. Usually, the transmittal form is checked only to see that the routine is put into the proper position in the family tree and to note any unusual requirements. Later, as the design matures, additions to the CCF are checked more closely to determine the reason for the addition or change. During system integration and test, changes are carefully reviewed-often by a Software Configuration Control Board (SCCB). After the Physical Configuration Audit (PCA) has been successfully performed prior to acceptance, additions would generally occur only as the result of an Engineering Change Proposal (ECP). The major point of control of the CCF is the program librarian, who is the only person who can update the file.

In the early stages of design and development, the identification of files and entry into the CCF is the only configuration management activity that takes place. As design and development progresses, code is written, compiled, and tested in private directories. At some point, the routine is ready to be tested with other routines in the system. To build an executable task, the analyst registers or places his routine in the system directory from which executable tasks are created. A routine called a Place Processor is used to transfer the routines from a private directory to the system directory. By executing standard Compiler and Catalog Processors, a load module is built that includes the routine of interest in addition to all other required routines that had been previously placed and compiled in the system directory. The Place, Compile, and Catalog Processors, besides performing their indicated tasks, make automatic entries in system journals or audit logs that indicate date, time, file name, and nature of the action performed. Logs of this nature answer questions such as, "When was a routine last compiled?" or "When was a task last cataloged?" These actions are performed in a private development directory so that the logs can also indicate the directory from which an action was requested and, therefore, indicate by implication who performed a particular action.

As development progresses and integration and testing is performed on a larger part of the system, there is an increasing possibility that two or more people may be working in the same area and making changes that may interfere with each other. This is prevented by implementing arbitration logic. The arbitration logic requires an analyst to reserve a file for his exclusive use any time that he is working on it. The analyst uses a Reserve Processor to copy the routine of interest from the system directory into his private directory. The Reserve Processor sets a flag in the CCF to note that the file is reserved and also notes who has reserved the file. If a second analyst attempts to reserve the file, the system will inform him that the file has been previously reserved and by whom. The second analyst then knows with whom to coordinate his change. When the work on the file is completed, the analyst uses a Return Processor to return the routine to the system directory and clear the reserve flag, which allows someone else to reserve the file for their use. Again, the Reserve and Return Processors make entries in the audit logs to leave a trail of which actions have been performed on the file.

Depending on the stage of development and the needs of the project, the amount of control to be exercised can be varied. Generally, up until system testing starts, little control is exercised and any analyst can reserve and make a change as required. If greater control is required, it can be accomplished by putting restrictions on the use of the Return Processor, i.e., by requiring authorization to return a routine and/or allowing only the program librarian to return the routine.

The next step, the release of a file to the configuration management system, usually occurs prior to the start of formal testing. This is a formal step that requires a review and authorization by at least project management. After authorization is given, a Release Processor is executed, which updates an audit log, marks the file in the CCF as released and generates a release record. Depending on how the system is implemented, the Release Processor may copy the file to a configuration manager-controlled directory or disc pack. The Release Processor is restricted to use by only the program librarian, who verifies that the proper authorization has been obtained.

If changes are required to be made to a released file, the assigned analyst can get a copy of the file into his private directory by using the Reserve Processor. Just as before, the processor marks the file for exclusive use by the requesting analyst. The file is modified, tested, and is eventually ready for rerelease. The program librarian rereleases the routine by using the Release Processor. In this case, the Release Processor notes the new release and revision information in the proper logs and release records.

The point of control at this stage is the program librarian, who is the only user of the Release Processor. This means that criteria can be established for release or rerelease of files. Typically, the Software Configuration Control Board of which the program librarian is a member reviews and authorizes changes to released software. Tighter control may be instituted by putting restrictions on the Reserve Processor by allowing only the program librarian to reserve released routines.

The program librarian plays an important part in the configuration management system described above. A few words about the qualifications of the person who fulfills that position are in order. The program librarian is assigned to the project team by an independent configuration management organization. The librarian's administrative supervisor is a configuration manager who is responsible to insure that configuration policies and directives are being carried out by the project.

The program librarian is responsible to see that configuration management procedures are carried out on a day-to-day basis. The librarian must be familiar with general configuration management

principles and have a thorough working knowledge in the use and operation of the procedures and processors described above. A basic computer literacy and an understanding of record and file struction is a necessity.

Summary

The system as described has several distinctive features. First, from the standpoint of the development team analysts, they perform few, if any, configuration management activities per se. For the most part, they are simply using processors to perform tasks that they would do in the normal course of development such as move files between directories, compile routines, catalog tasks, etc. As a by-product, the processors create audit log entries that any analyst can access when they have questions as to when or who performed certain actions.

Second, the information kept by the system is up-to-date and immediately available. Through an audit log interrogation processor, the system informs project personnel who and when various activities were performed.

Third, the system performs a service in that it protects the analyst from concurrent interacting changes that others may implement. Further, if the analyst tries to make a change on a file that someone else is working on, the system will tell him who has the files.

Fourth, the level of control can be varied as the project or situation requires. In the early design phase, the system is relatively open, providing only a central file to keep data and report generators to access the data from the file. During code generation, the system provides standard tools to perform compiles and catalogs and keeps records of when they were used. Later, during integration, it arbitrates among users who may be trying to modify the same file. Lastly, it can provide full lockstep control that one would expect from a traditional configuration management system once design and development are complete.

Finally, since the configuration management system is resident on the host development computer, a high degree of operational efficiency is achieved. The data is collected and the status is available to the group that needs the data on the system that is being used for development.

Because this type of configuration management system provides a real benefit to the development team early in the design stage, a configuration management program similar to it is likely to be started early in the project. This, in turn, provides a high degree of visibility in the early crucial stages of the design phase. Standard processors for system development tasks improves efficiency and encourages coordination among project personnel. Through the generation of logs, an accurate history is kept of key development activities so that traceability between what was specified and what was built can be verified.

Acknowledgment

Much of the configuration management experience presented in this paper was gained on contracts sponsored by the: Air Force Systems Command, Aeronautical Systems Division, Attn: ASD/YWKB, Wright-Patterson AFB, OH 45433.

About the Author

Mr. Michael Moriarity is a Senior Design Analyst in the Electronic Warfare Operation at AAI Corporation. He is currently the Software Manager on the EF-111A Operational Flight Trainer Program. Mr. Moriarity holds a Bachelor of Science degree in Mathematics from the University of Minnesota and a Masters of Engineeirng Administration from George Washington University. Previous to his current position, he was responsible for software development on the Navy Electronic Warfare Trainer System (NEWTS) and the defensive instructional subsystem on the B-52 Weapons Systems Trainer. He has also had extensive experience in developing software for automatic test systems.