Technical Report

CMU/SEI-90-TR-25
ESD-TR-90-226

Tool Version Management Technology:

A Case Study

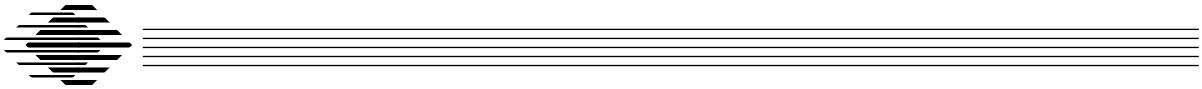
Peter H. Feiler
Grace F. Downey

November 1990

Technical Report

CMU/SEI-90-TR-25
ESD-90-TR-226
November 1990

Tool Version Management Technology:
A Case Study



Peter H. Feiler

Grace F. Downey

Software Development Environments Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the SEI Joint Program Office HQ ESC/AXS

5 Eglin Street

Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF, SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Research Access, Inc. / 800 Vinial Street / Pittsburgh, PA 15212. Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page at http://www.rai.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service / U.S. Department of Commerce / Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides acess to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential con tractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218. Phone: 1-800-225-3842 or 703-767-8222.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Tool Version Management Technology: A Case Study

Abstract: This report describes a portion of the problem of maintaining tools for the purpose of software development. It discusses an innovative solution to tool version management available in the commercially available Network Software Environment from Sun Microsystems, Inc. It applies NSE mechanisms to solve three problems that are common to the use and management of tools for software development.

1. Introduction

This report discusses the results of the tool version management task, which is one of several tasks being done as part of the Software Development Environments Project at the Software Engineering Institute. The primary focus of this task is on environment support for the management of tool versions and automation of tool version selection. The purpose of this document is to describe the problem of tool version management and selection, and to discuss an innovative solution available in a commercial software development environment.

First, we discuss a problem that any facility providing support for software development faces: the management of software development tools. The problem is broken down into three issues: tool version organization and selection, the stability of selected tool versions and the application context of selected tool versions.

Sun Microsystems' Network Software Environment (NSE) provides mechanisms that address not only software source and object management but also tool management. NSE is designed foremost to address the problem of managing configurations of source code and derived objects. Tools are managed as the producers of the derived objects.

Chapter 2 discusses some specific problems in the tool management domain. Chapter 3 is provided to outline the key features of Version 1.2 of NSE which are pertinent to its tool management capabilities. Chapter 4 details how tools must be organized to capitalize on NSE's capabilities and then shows three areas where the NSE mechanisms are useful: development in heterogeneous networks, managing tool releases and organizing task-specific tool sets. Chapter 5 explains some direct and indirect mechanisms that NSE provides to aid in parameterization management. Chapter 6 describes how NSE's tool management solution can be used to capture tool application context. Chapter 7 contains our conclusions.

2. The Problem

It is now common for software development environments (SDEs) to consist of a number of tools [4]. These tools operate on data that evolves over time and whose history is recorded through a version management tool. Different tools work on different data and at any one time only a subset of the tools needs to be accessed. Tools do not exist as a single instance. Variants may coexist to produce objects for execution on different hardware, and to reflect adaptations and tailoring for project and task-specific needs. Multiple and incompatible releases may exist and may need to be available at the same time to work on new and older versions of data. Thus, an SDE has to deal with a proliferation of tool versions. This raises a number of issues regarding tool version organization and selection, regarding the stability of the selected tool configuration in light of the presence and change of other tools and tool versions, and regarding the context in which the selected tool version is applied.

2.1. Tool Version Organization and Selection

Versions of tools can be organized according to several different criteria. This creates a multi-dimensional version space. Typical dimensions are machine architecture, operating system and window system type and version, and tool release sequence. Some tools account for parameterization of certain dimensions while others do not, requiring system administrators to develop a range of installation conventions.

The organization of tool versions has an impact on tool version selection. Tools are invoked by a number of agents, ranging from the user to build scripts, menu systems, notification mechanisms, and other tools. Thus, tool version information is distributed in a number of places. Uniformity in tool version organization and localization of version selection reduces complexity.

Selection of tools and tool versions is context sensitive. Tools can be selected based on the type of data to be operated on and the task to be performed. Tool versions are repeatedly selected based on their invocation context, e.g., tool variants for particular machine architectures. Therefore, it is desirable for an environment framework, i.e., the platform into which tools are integrated, to provide an automated tool version selection mechanism, resulting in consistency of repetitive selection of tool versions.

2.2. Stability of Selected Tool Versions

Many tool versions are generated by parameterization of a tool base configuration. The resulting tool version is a refined tool configuration consisting of a base configuration and a collection of parameter values, which may be stored in a number of locations. A number of parameterization techniques are available to provide the flexibility of adapting a tool without complete replication. They use dynamic binding of parameters at invocation and execution time.

Many of the parameterization mechanisms apply globally and do not support the notion of scopes. Different tools and tool versions have to share these mechanisms. The result is that parameterization of one tool may unintentionally change the configuration of other tools or tool versions, i.e., the stability of any tool version potentially can be affected.

2.3. Application Context of Selected Tool Versions

Tools operate on data. Some tools create and modify source data. These tools produce new versions of essentially the same data. Other tools process data as input and generate new versions of output. While source data is key to the resulting output, Figure 2-1 illustrates three additional factors that impact the operation of tools and the tool output.

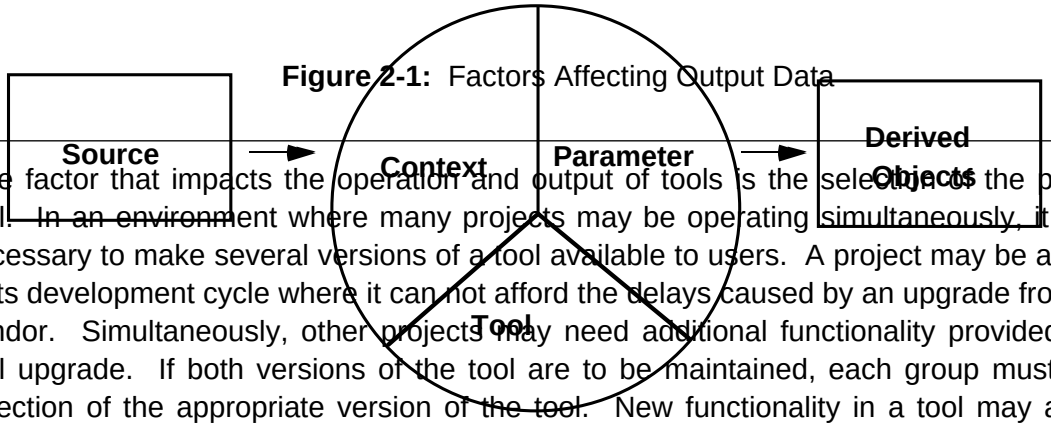


Figure 2-1: Factors Affecting Output Data

One factor that impacts the operation and output of tools is the selection of the particular tool. In an environment where many projects may be operating simultaneously, it is often necessary to make several versions of a tool available to users. A project may be at a point in its development cycle where it can not afford the delays caused by an upgrade from a tool vendor. Simultaneously, other projects may need additional functionality provided by the tool upgrade. If both versions of the tool are to be maintained, each group must ensure selection of the appropriate version of the tool. New functionality in a tool may affect its operation and an upgrade to the tool may affect the nature of the format or content of the tool's output.

The second factor is parameters passed to the tool. Some parameters passed to tools are refinements of a tool configuration and effectively accomplish tool version selection. Other parameters are specifically passed to the tool to make a change in the derivation process. The latter parameters are usually recorded as part of the build description of a software product, reflecting versions of the product rather than versions of tools. This indicates that there is a strong connection between version management of tools and version management of software products.

The third factor that has impact on the operation of tools is the context in which the tool executes and in which the data resides. Tools sometimes make use of facilities assumed to be available as part of the computing environment they are installed on. On UNIX a typical example is any tool that makes use of the pattern match program *grep*. Such programs are usually not part of the release configuration that a tool vendor supplies. Tools usually operate on data in the context of other data. Some of that data is part of what is being produced. Other data is often assumed to be available in the computing environment, e.g., system libraries and interfaces. In order to guarantee consistent results, stability of the application context of tools must be maintained.

3. Network Software Environment (NSE) Technology

The Network Software Environment, commercially available and supported by Sun Microsystems, Inc., provides software evolution support that is useful to a single developer as well as useful to coordinate the work of a group of developers [3]. This report is based on our experience with Version 1.2 of NSE. NSE has been chosen for discussion in this report for two reasons. First, it is a system that addresses the problem of tool versions and offers an innovative solution. Second, advances have been made in software configuration management (SCM) capabilities in support of software development. Sun NSE combines a number of them to offer SCM in a non-intrusive manner to developers in UNIX environments. Tool version management support is offered in the context of the NSE SCM capabilities. A factor that is most influential in determining the version of tool output is the version of input to the tool. In Section 3.1 configuration management is discussed because it supplies the input to the tool. When input may consist of a configuration of more than one file, a tool may be considered to operate in a context as opposed to operating on a particular source file. Section 3.2 describes the transparent source version selection mechanism that is also used to provide tool selection. The mechanism sets up the context in which the tool operates. NSE provides derived object management such that more than one set of derived objects may be associated with a given set of sources. It is often the case that different derived objects are the result of the application of different versions of tools. By having a mechanism to support objects derived by tool variants, an avenue is opened to manage the selection of the appropriate tools. Section 3.3 explains how the mapping to the correct derived objects is provided. Sections 3.4 and 3.5 describe how tools are mapped into the context provided by NSE's data management capabilities. Section 3.6 describes how NSE captures environment variables as part of the context in which a tool operates.

3.1. Source Configuration Management

The key software evolution support concept in NSE is an *environment* [2]. As a software system is created and maintained, developers are constantly changing source files and recompiling them to produce new derived objects. An environment is a managed object which in turn contains the objects used to construct a software system. The environment provides a workspace in which to control and coordinate the change occurring to the software system. A given environment may contain configurations of source files, derived objects and UNIX directories. The particular version of files found in an environment provides a context in which a developer and software development tools operate.

As the contents of an environment evolve through development and maintenance, stages in the development may be recorded. The *preserve* command takes a "snapshot" of the contents of an environment for later retrieval. This capability of environments allows them to contain a linear version history for a configuration of files, derived objects, and directories. Figure 3-1 depicts a UNIX directory structure containing sets of sources and derived objects moving through three revisions. The outermost box containing revisions R1, R2, and R3 represents an NSE *environment*. For storage optimization, NSE copies only objects that have changed from one configuration version to the next. These objects are shown in Figure 3-1 as the small shaded boxes. NSE's specialized file servers make the configuration appear complete and consistent to a user accessing it (see Section 3.2).

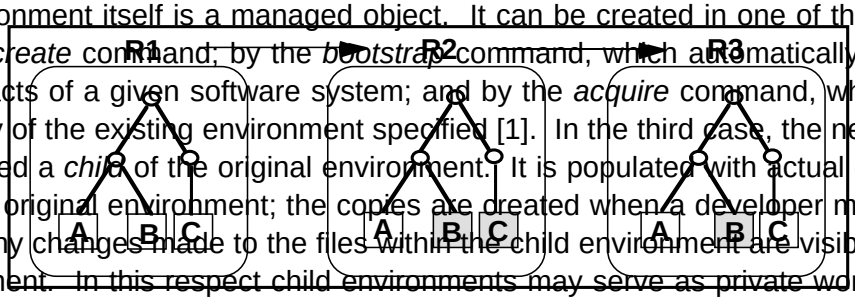


Figure 3-1: Linear Version History of a Configuration

An environment itself is a managed object. It can be created in one of three ways: by the *nseenv create* command; by the *bootstrap* command, which automatically populates it with the artifacts of a given software system; and by the *acquire* command, which creates a virtual copy of the existing environment specified [1]. In the third case, the new environment is considered a *child* of the original environment. It is populated with actual copies of the files from the original environment; the copies are created when a developer makes a change to a file. Any changes made to the files within the child environment are visible only within that environment. In this respect child environments may serve as private workspaces. After a developer has completed changes, and tested them for accuracy, the changes may be returned to the original environment or *parent* via the *reconcile* command.

Figure 3-2 shows the sequence of child environment creation, change, and propagation of change back to the parent environment. This represents a transaction style software configuration management model. The transaction model, as implemented by Sun's NSE, supports the creation and coordination of workspaces for concurrent software change with an automated merge procedure.

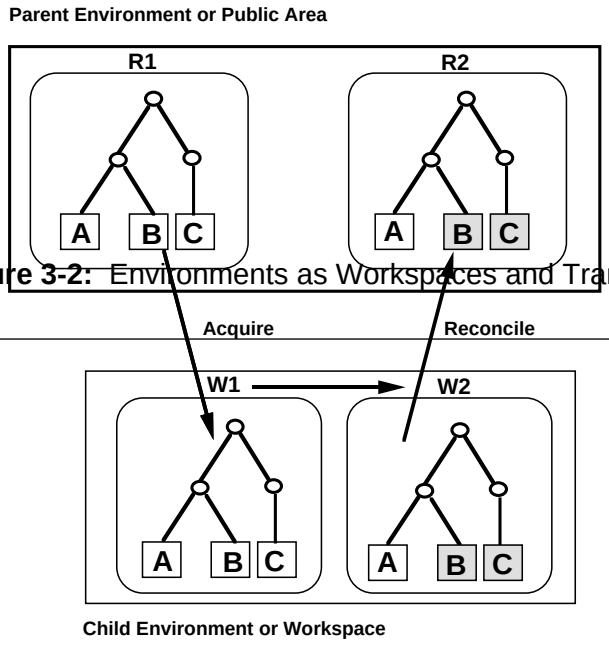
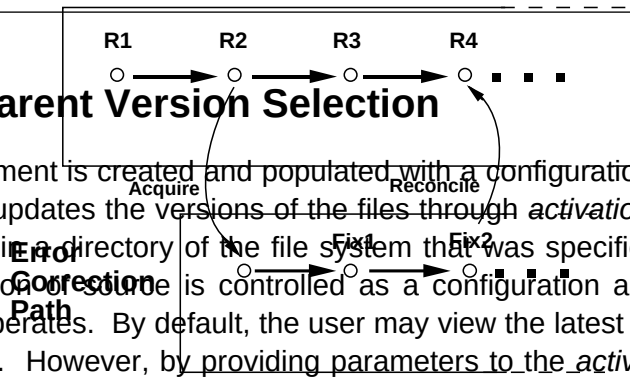


Figure 3-2: Environments as Workspaces and Transactions

An NSE environment can be used to maintain separate development paths for a software system. Often during software development, it is necessary to maintain two versions of a software system simultaneously. A typical situation occurs when an organization supports a released version of the software to make rapid fixes to errors reported by customers. At the same time functionality enhancements may occur to a separate copy in preparation for the next major customer release. Each of these versions of the software system can be called a *development path*. NSE environments can be used to manage a software system which splits into separate development paths. As the software system progresses through successive changes in a given environment, it has a single development path. At the time of system release, a child environment can be created to provide a place to test and fix error reports, as shown in Figure 3-3. The successive upgrades occurring in the child environment represent the second development path.

Figure 3-3: Environments as Development Paths



3.2. Transparent Version Selection

After an environment is created and populated with a configuration of files and directories, a user views and updates the versions of the files through *activation* of the environment. The files will appear in a directory of the file system that was specified at environment creation time. The version of source is controlled as a configuration and provides the context in which the tool operates. By default, the user may view the latest configuration maintained in the environment. However, by providing parameters to the *activate* command, a user may view earlier *preserved* configurations. In function, environments provide the ability to view a version of a configuration in a transparent manner.

The contents of an environment are actually stored in a repository which is built in the UNIX file system and controlled by NSE. NSE provides a specialized file system server which provides access to the correct versions of files as they reside in the internal structure. An

activation starts the specialized file system server as a separate UNIX process. Within the process or *activated environment*, the specialized file system server will make the correct versions of the files and directory structure appear in the designated directory. The designated directory was specified at the time the environment was created and is called the environment's *control point*.

By providing a *per process mount* of a configuration of directories and files, different processes may simultaneously access different versions of the configuration of files. Figure 3-4 shows two developers and their tools accessing two versions of the configuration simultaneously. The different file versions appear to each developer at the same control point in the file system.

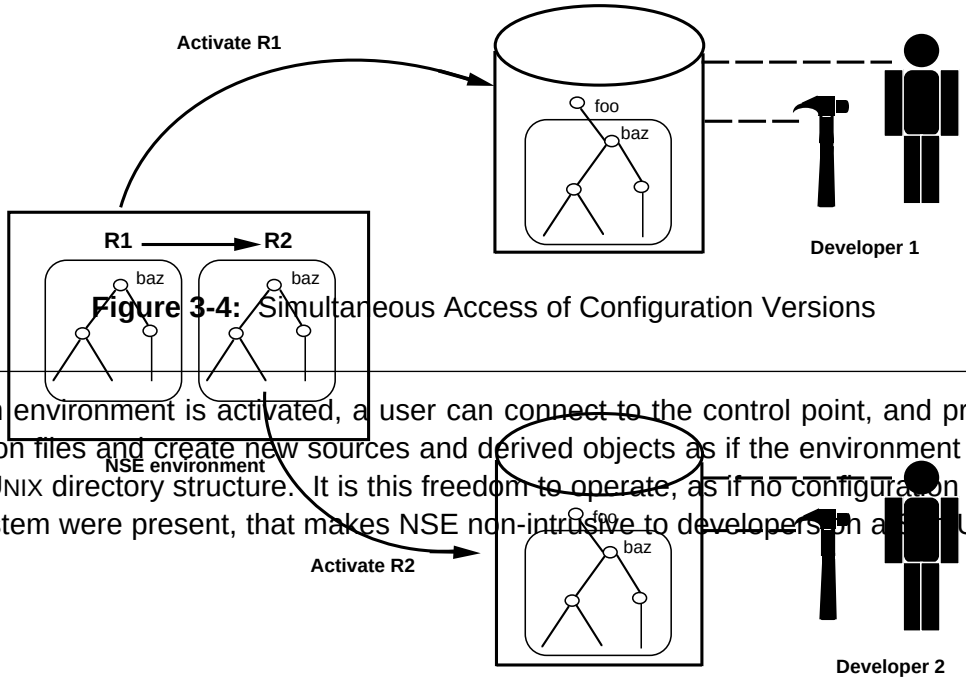


Figure 3-4: Simultaneous Access of Configuration Versions

When an environment is activated, a user can connect to the control point, and proceed to operate on files and create new sources and derived objects as if the environment were the normal UNIX directory structure. It is this freedom to operate, as if no configuration management system were present, that makes NSE non-intrusive to developers on a UNIX platform.

3.3. Derived Object Management

In addition to supporting multiple versions of source objects, NSE provides the ability to maintain several versions of derived objects. Often a set of sources is processed by different tool variants to generate multiple sets of derived objects. For example, several different compilers are used to generate sets of derived objects, or switches passed to a compiler on different builds result in different objects and executables. NSE maintains multiple derived object sets by providing the ability to associate all the different sets with one set of sources. An NSE environment can contain a version of a set of software sources and several different *variants* of derived objects. Figure 3-5 depicts how parameters in the *activate* or *acquire* commands specify which set of derived objects is seen by users with the sources. Within an activated environment, a tool such as *make* or the compiler will operate on a consistent set of source and derived objects. Derived objects produced by a different compiler in a separate variant will not be visible. The same mechanism that supports the selection of derived objects from a set also makes it possible to select the object deriver or tool.

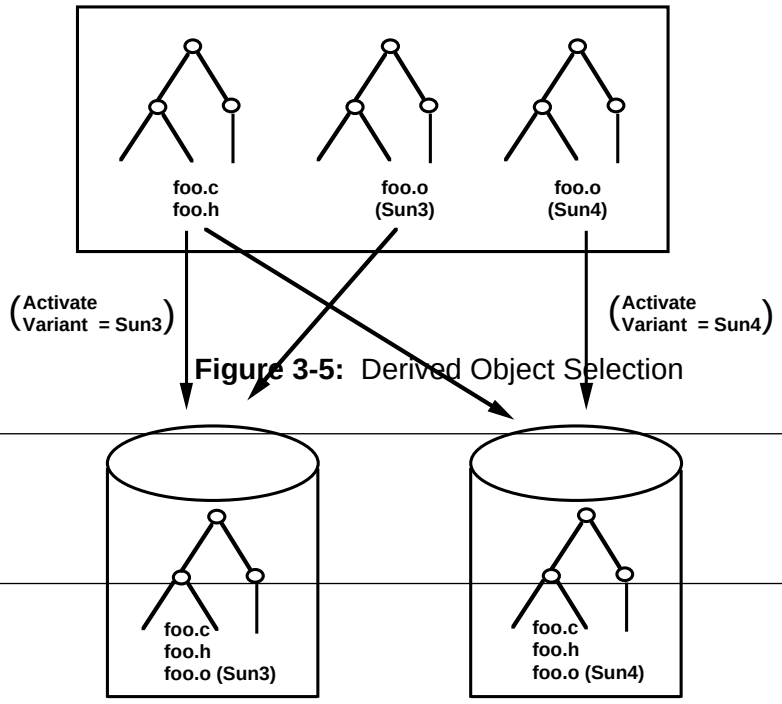


Figure 3-5: Derived Object Selection

3.4. Tool Sets

Whether an environment is used to retrieve a particular configuration of source and variant of derived objects, as a workspace for an individual developer, or to house a particular development path, it provides a working context for developers and the operation of tools. The Network Software Environment *execset* provides the ability to designate a tool set that is appropriate for a given environment. Under UNIX the availability of a tool for a user is determined by the user adding the tool's directory to his search path. NSE provides more control over access by allowing automated selection of a particular version of a tool for a given environment. With the use of execsets, neither build utilities, such as *make*, nor users have to be concerned with selection of the correct tool version.

An execset is a list of directories containing tool executables, such as a compiler or debugger. An execset may also contain files which are needed for correct compilation, such as the system libraries containing standard input or output routines. The tools and files placed in an execset are accessed by transparently mapping their execset directories on top of the native file system directories where other versions may be found. For this transparent mapping, NSE uses the same per process mount capability as is used for making source and derived files of the environment transparently accessible.

It is possible to make the layering of execset directories either *opaque* or *translucent.* If the directory specified in the execset is designated as *opaque*, when the execset is activated it will block out all of the contents of the corresponding native file system directory. Only the executables and files installed in the execset will be available from the native file system directory. An opaque execset directory would be good to use in the case where one directory contains all the executables that belong to only one tool. It could be blocked out in its entirety to be replaced by the complete set of executables of the later version of the tool. If the directory is *translucent*, then the tools found in the execset directory will be used in place of the tools on the native file system. However, the rest of the files present in the native file system directory can still be accessed by the user and tools. A translucent execset directory would be good to use in the case of /usr/bin. For example: if an alternative version of *cc* were desired, but other tools such as */usr/bin/awk* or */usr/bin/lint* were still useful, then */usr/bin* should be added to an execset and designated as *translucent*. The alternate version of *cc* would be installed in the execset */usr/bin* directory. When the execset is active and a user invokes *cc*, NSE's specialized file server would cause the invocation of the alternate version of cc. The original version of *cc* would be obscured. However, the rest of the contents of */usr/bin* could still be invoked as if the execset were not active.

An execset is associated with a particular environment by assigning the execset to the environment with the *nseenv execset* command. The execset will remain associated with the environment until it is replaced with a subsequent *nseenv execset* command or a remove execset command (*nseenv rmexecset*.) By removing the attachment of the execset from the environment, tools as they exist in the native file system are available within the active environment again. Tools are placed in the execset by moving them into the directory indicated by the *nseexecset prefix* command. The directory that it indicates is part of the NSE

maintained storage structure. Once the execset has been attached and populated with tools, any time the environment is activated, the tools of the execset are automatically available for use. Only one execset set may be designated at a time for an environment.

A given set of tools installed in one execset's directories can be associated with multiple environments. For example, if a syntax-directed editor, compiler, debugger, and performance analysis tool worked well as a package, they could be placed together in one execset directory. All four tools could be made available as a package to many different environments by associating the execset with each environment as needed. It is also possible for a popular tool to be made available for use in many environments by placing it in many execsets. Figure 3-6 illustrates the possible mappings of environments to execsets to tools.

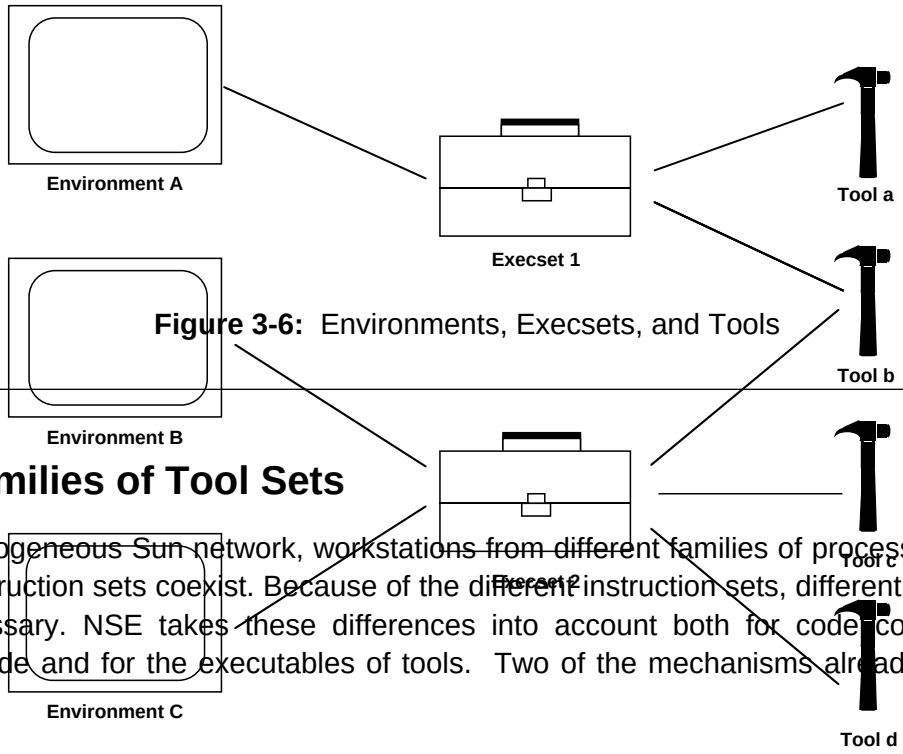


Figure 3-6: Environments, Execsets, and Tools

3.5. Families of Tool Sets

In a heterogeneous Sun network, workstations from different families of processors with different instruction sets coexist. Because of the different instruction sets, different executables are necessary. NSE takes these differences into account both for code compiled from source code and for the executables of tools. Two of the mechanisms already described,

variants and execsets, can be used in conjunction to manage the development of code on and for a heterogeneous Sun network.

NSE uses *variants* to attach several sets of derived code to the same source. Typically, variants arise from derived objects that are generated for the three Sun machine architectures (as indicated by the operation */bin/arch*). Execsets may be characterized by the host and target architectures they are intended for. For example, in a mixed Sun3 and Sun4 network, there may be a separate execset for tools executing on Sun3s generating code for Sun3s, tools executing on Sun3s cross-compiling to Sun4s, and the equivalent tools executing on Sun4s (as illustrated in Figure 3-7). Such groups of execsets are referred to as *execset families*.

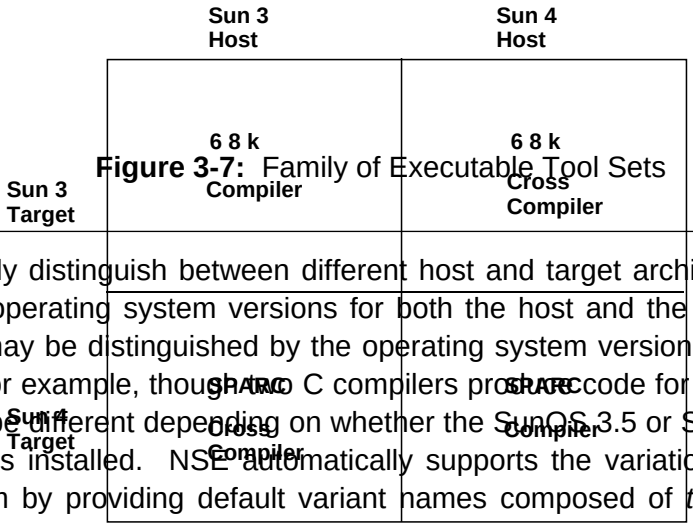


Figure 3-7: Family of Executable Tool Sets

Execsets not only distinguish between different host and target architectures, but also between different operating system versions for both the host and the target. Different versions of a tool may be distinguished by the operating system version with which they must be executed. For example, though two C compilers produce code for Sun 3 computers, the compilers must be different depending on whether the SunOS 3.5 or SunOS 4.0.3 operating system version is installed. NSE automatically supports the variation of architecture and operating system by providing default variant names composed of *target-target operating system.*

NSE's environments, variants, and execsets provide a powerful mechanism to control software development targeted at different hardware platforms and operating systems. If

the environment is set up properly, a user is able to activate the environment for a given hardware target, automatically view an appropriate version of sources and derived objects at a designated point in the file system, and receive an appropriate version of tools.

3.6. Parameterization Management

Environments serve as a meeting place where a selected version of data objects may be operated on by a selected version of tools. Just as NSE's specialized file server provides an appropriate version of data and tools with the activation of an environment, so too does the file server store and automatically redefine the value of environment variables. The user may invoke the *nsevar assign* command to assign a designated value to a UNIX shell environment variable for the duration of the activation of the environment. When the environment is de-activated with the *exit* command, environment variables will assume the values from before the activation of the environment. Upon subsequent activations of the environment, the variable will be automatically reset to the designated value.

4. Tool Version Organization and Selection

The previous section described the mechanism that NSE provides to select versions of source, variants of derived objects, and versions of tools and to capture environment variable values. This section elaborates on some techniques for organizing tool installation to allow a maximum benefit from the NSE mechanisms. Through a combination of mechanism and development policy, it is possible to select tools and manage derived objects for development in a heterogeneous network. The mechanisms may also be used to ease the transition from one version of a tool to successive versions when the releases are incompatible. Finally, it is possible to organize tools into sets according to specific tasks.

4.1. Effective Tool Organization

Two aspects of the creation and use of an execset must be considered for effective tool organization: the portions of the native file system to be masked and the placement of tools into execsets. The *nseexecset create* command uses a file that lists the directories where tools appear in the native file system. A directory listed in the file followed by an "O" (as in "opaque") will contain only tools specifically added to the execset. The contents of the execset completely overlay what may appear under the native UNIX file system for that directory. This can be done to completely replace the contents of a directory during the activation of an environment. It can also be used to simply block access to portions of the file system. If a particular directory contains too many valuable tools to be completely layered over, then when an execset is created, the directory can be listed in the file followed by a "T" (as in "translucent"). In this case, when the execset is active, the tools found in the native UNIX file system will still appear. Only those tools added specifically to the execset will overlay the particular tool from the native file system.

It is also possible to create empty directories in the native file system, and execsets can be used to "layer in" appropriate versions of tools. The empty directory must be added to a potential user's search path. This can be done by adding the directory to the PATH variable, which is often defined during the user's login sequence. Searching an extra directory for executables could also be confined to the period when an environment is activated. Since PATH is an environment variable, the *nsevar assign* command could be set up to redefine PATH to include the tool directory.

A tool becomes available in an execset after the user has installed it there. The *nseexecset create* or *nseexecset set* command must be invoked first to specify the execset to which the tool will be added. The right column of Figure 3-6 depicts Tools *a* through *d* as they might reside on the native UNIX file system. Tools can be placed in an execset (depicted as tool boxes in the middle column) by making a physical copy into the NSE internal data structure that the *nseexecset prefix* command indicates. Another method is to have tools in a designated tools directory that will not be overlayed by execset directories. A symbolic link may be established from the execset directory to the actual location and executable instance of the tool.

If a tool is required in more than one execset, it may be copied into each execset. If the tool is large, in order to save the disk space needed by copies, the tool may be accessed via a symbolic link to one instantiation as described above. It would be wise to treat the tools in a designated tools area as immutable in this case. Users and environments may rely on a particular version of a tool via a link from an attached executable set. If the tool is changed or replaced in the designated tools area, it is difficult to establish at the time who had links pointing to it. If users can rely on a tool being immutable, then there will be no need for multiple copies to ensure that a particular build context has not changed.

4.2. Tools and Networks

With the use of workstations and network file systems, tools are operating in a different context than was provided by mainframes with no processor access terminals. Sun's Network File System makes it possible for more than one type of workstation architecture to reside on the same network and share file space. Development must also manage the creation of several versions of a product that will execute on different workstations, and appear similar in function across the different processors. This section describes how NSE *execsets* may be used to organize tools in this situation. The situation is more complex because of the variety of workstations offered by vendors. The availability of Sun's Network File System protocols makes it possible for more than one manufacturer's equipment to reside on the same network. Section 4.2.2 discusses the limitations of NSE in a network with a mixture of hardware from different vendors.

4.2.1. Sun Networks

NSE *variants* can help to maintain a single source, and a set of tools organized in an *execset family* may be used to produce alternative executable versions for different processors. With appropriate naming schemes, NSE *environments* may be set up as workspaces which automatically give the developer access to the correct version of source, tools, and derived objects.

When an environment is activated, the derived files for one variant are made accessible under the control point, together with the source files. By default, the variant is that of the architecture of the workstation on which the user is activating the environment. However, the user can specify an alternative variant at activation time. This allows a user to reside on a machine of one Sun family, i.e., the *host*, and build code for a second Sun family, i.e., the *target*, through cross compilation. The executable code, however, can be invoked and tested only on a machine that is a member of the target family. This is done by activating the environment on a machine of the target architecture. By default, the sources and the executables appropriate for the target will be transparently accessible on the target machine.

When creating an environment with the *bootstrap* command or the *nseenv create* command, NSE labels the environment with the host architecture and operating system. NSE assumes that the environment will produce derived objects for the given architecture and operating

system. However, additional variants may be specified with the *bootstrap* and *nseenv create* commands, labeling them with the name of the target architecture and target operating system on which derived objects will execute. If an environment is created on a Sun3 running operating system version 4.0.3, it may also have a *Sun4-2.0.1* variant. This variant would contain object code that would execute on a Sun4 processor with operating system version 2.0.1. NSE uses the target architecture and operating system version number present in the variant name to select the appropriate execset from the execset family. When the execset name, host and target information are given as parameters to the *nseexecset prefix* command, the command returns a path to an NSE managed directory. Tools that perform work for the indicated host and target can be installed into the indicated directory. Then the execset family may be attached to an environment through the *nseenv execset* command. The result of the careful naming and tool installation is that two views of the same environment will share source, but create target-specific derived objects. When a developer activates an environment and specifies the desired variant, NSE will automatically provide the necessary tools to work in the environment. Only one build script or *Makefile* needs to be maintained for all variants of the system, as tools residing in different sections of the execset family can have the same name. For example, if two versions of a C compiler *cc* exist, one to produce code for a Sun3 system and the other to produce code for a Sun4 system, each version of *cc* could be installed as *cc*, but in different members of an execset family. Then the build script may invoke a compiler as *cc*, and under the Sun3-4.0.3 variant of the environment, the version of *cc* that produces object code for the Sun3 architecture and 4.0.3 operating system is invoked. However, the same build script may be used under the Sun4-2.0.1 variant of the environment, and will automatically invoke the version of *cc* that produces object code for the Sun4 architecture running the 2.0.1 operating system.

Using environment variants, execset families, and properly installed tools, it is possible to maintain only one set of sources for all versions of the product running on different platforms. Upgrades and error correction can then be easily synchronized across all platforms. A developer need only specify an environment and its variant to access correct versions of the source, tools, and derived objects.

4.2.2. Heterogeneous Networks

Because NSE is currently available only on Sun workstations, development on machines from other manufacturers (such as Dec VAX ULTRIX or VMS, Hewlitt-Packard, etc.) can be supported only to various degrees. The problem with lack of availability of NSE on non-Sun machines is that files in an activated environment are not directly accessible, although those machines may be connected to Suns through a network and have access to files via the Network File System (NFS). One reason is that the NFS server providing files for non-Sun machines is not aware of the contents of an activated environment. Remote access to the contents of an activated environment is not currently available. One of the key elements to allowing remote access is the ability to mount file systems on a per process basis [5].

An additional limitation is that the tools necessary to cross from one manufacturer's platform to another are uncommon. For example, it may require in-house development to produce a compiler that will run on a Sun3, yet generate code for a Dec VAX ULTRIX system.

4.3. Incompatible Tool Releases

Often, tools are not static objects, but tend to evolve through a series of revisions. To discuss how NSE can manage the upgrade of tools, we divide the problem into three areas based on the magnitude of the tool change:

- tool internal
- output format
- input and output format

A tool change is designated as *tool internal* when it has no impact on format of input data or output data. The output of the new tool is compatible with the output of the former version. An example of a tool internal change would be an ANSII C compiler upgrade for faster compilation speed that did not affect the object code it produced. If it has been the policy to simply copy needed tools into execsets; then the new tool may be simply copied over the old in each execset to which it belongs. However, it is most prudent to create a new execset by making a copy of the original execset. The new execset can retain all the references to other tools but be updated to reference the upgraded tool. Then environment users can switch to the new version of the execset but may still revert to the old if necessary. Even after all users have upgraded to the new tool, it will be necessary to maintain the old tool if exact version recreation is needed. The simplicity of this type of tool upgrade rests entirely upon the nature of the change in the tool.

It is rare when a tool upgrade is as benign as described in the previous paragraph. Often there is no need for the input to the tool to change, but the *output format* of the tool has changed. If an ANSII C compiler is upgraded to take advantage of a new calling sequence, the C source need not change, but a completely new variant of derived objects is required. In this instance, the output of the two versions of the compiler would not be compatible. For some applications, it may not be clear whether to take advantage of an upgrade. NSE variants provide a good way to create and manage both sets of derived objects. The user can add a new variant to the environment that would reflect the output of the new tool version. An executable set could be created retaining the standard set of tools with the upgraded tool. When the environment is activated with the new variant, the new set of tools is made available. In this environment and variant, the derived objects can be rebuilt and tested. If the source can continue to remain the same, development on the two variants may even continue in parallel. Changes in source are automatically seen in activations of the environments, regardless of which variant the source is appearing with.

A tool is completely incompatible with earlier releases if both the *input and output format*

change between releases. A different use of execsets and environments can help to manage the upgrade to the new tool version. The new version of the tool should be placed in a new execset. In order to maintain the reproducibility of a software system, the new execset should not just be attached to all environments using the tool. Rather, an environment that must upgrade to the new tool should spawn a child environment through the *acquire* command. The child environment can then undergo a one-way conversion from the old input format for the tool to the new input format. The execset containing the new version of the tool can be attached to the child environment, and the system can be rebuilt. The parent environment will still exist with the old format, and will still invoke the old version of the tool. Even though a child environment may be used to house a separate branch of development, in this case it may serve to merely continue development. The parent environment provides the ability to reproduce past versions of the system with out-of-date tools.

4.4. Task-Specific Tool Sets

It is possible to attach sets of task-specific tools through the execset mechanism to the data stored in an NSE *environment.* The data placed in an NSE *environment* is accessed through the *activate* command with parameters specifying the name of the environment and possibly also the revision number and variant. Since an execset may be set up to block out part of the file system, and only the tools specifically placed in the execset are present, a developer may be limited to a pre-designated set of tools for a given environment. For example, documentation specialists may be prevented from invoking a compiler, but text editors and text processing systems are made available in the tools directory through the active environment and execset. The use of execsets in this manner has the same effect as a menu system that restricts the access to tools by providing only "start buttons" for a pre-designated set of tools.

Different tool sets may be accumulated under different execset names. This builds to the concept of a *typed* environment, which limits the developer's access to only a specific kind of development object through the NSE *environment*, and limits the actions that may be performed on those objects through the NSE *execset.* An NSE *environment* with an *execset* can come to represent a specific development task by virtue of its type. The task of documentation production may have an environment with access to only documents and editing and formatting tools. The task of integration may have access to only the module interfaces and appropriate editor by virtue of the environment activated. Documentation, code, and interface code may all reside in the same UNIX directory structure, and it is through the environment and execset mechanism that task restrictions become possible.

It is possible that the same tool may be used across two different tasks. However, the nature of the tool and task require task-specific tool variation. Different NSE execsets may be set up to contain the different variations of the tool. A task-specific environment, can attach the appropriate execset. Picking up the appropriate tools will then occur as a function of activating the correct environment for the task. As an example, if a project produces code in both machine code and C, a given editor may be configured at installation time to provide C

syntax checking or machine code formatting. Two versions of the editor may be created, one installed in the execset for C programming use, and the other installed in the execset for machine code programming. Environments can be created and named *C-coding* and *machine-coding* with the appropriate execset attached. A given developer need not remember which editor resides where in the file system, but can activate the appropriate environment and invoke the editor under the same name for either case.

Within an execset there are still many of the same concerns that are present when installing collections of tools in the UNIX file system. An environment is limited to having only one execset active at a time. Within that execset, tools must be compatible. An execset is flat; that is, there is no support for collections of tools, each composed of several components. Related tools may still conflict unless placed in separate execsets.

5. Stability of Tool Configurations

The behavior and output of a tool is often affected by the value of parameters. NSE provides some capability to prevent the fluctuation of a tool's performance based on inadvertently changed parameter values from one tool invocation to the next. There are three common methods to provide parameters to tools: profiles stored in files external to the tool, command line option values, and environment variables. The following paragraphs discuss NSE's abilities to manage each type of parameterization and to contribute to tool stability.

If a tool relies on external files existing in the native file system to contain parameter values, then the external file may be placed in an execset along with the tool. A common example of an external file containing profile values are the files that a developer will set up to customize a window management system on a workstation. If the profile is expected to reside in a particular directory, the directory may be specified at execset creation time for either opaque or translucent access. In either case, the execset can then contain a version of the parameter file to overlay one which may exist in the native file system. As a member of the execset, this parameter file can remain constant across one activation of the NSE environment to the next.

If two tools rely on the same external profile file, then the operations of the two tools will interfere with each other if they require different values present in the file. The NSE execset mechanism provides a means to limit profile clashes. The execset directory can be used to block out the occurrence of the profile file in the native file system. Then, along with the installation of one of the tools in the execset directory, the profile with the appropriate values may also be installed. A second execset may be created that will contain the other tool, along with its version of the profile file. The values found in a profile will be used only within an NSE environment that is active and whose execset contains the profile file. When an environment is de-activated and re-activated, it will automatically attach to the same tool and profile as before. By providing a scope to parameters contained in profiles, NSE execsets and environments may be used to provide an easily replicated workspace which may choose to ignore site and user specific profiles. However, NSE toolsets cannot contain two tools with clashing profiles to be used simultaneously on the same environment. Execsets provide only one flat layer possible over the native file system.

The second method to provide parameter values to tools is through command line option values. An example of a parameter value that changes tool output is the *-t* option for the command *ls*. Use of this option will produce a listing of the files in a UNIX directory in time order. NSE allows for the tracking of command line options by using the Sun UNIX Make facility for tool invocation. The Makefile serves as a build script. The Makefile may have versions which are managed as part of the source configuration that is maintained in an environment. Also, through some NSE-specific enhancements to the Make facility, changes in these command line parameters are tracked as having an influence on derived objects. Upgrades to execsets or the entire replacement of a tool, however, will not cause a rebuild by the Make facility, and still require the user to force a complete rebuild.

The third method of providing parameters to tools is through environment variables. An example of the third method is a compiler that checks if the environment variable DEBUG is set. Based on the value of DEBUG, the compiler may insert debugger commands in the output code and not perform some optimizations. The *nsevar assign* command will assign a designated value to a UNIX shell environment variable for the duration of activation of the environment. This will overwrite any value of the environment variable that may be present from before the activation of the environment. If the environment is de-activated with the *exit* command, upon subsequent activations the environment variable value will be automatically set to the value designated in the *nsevar assign* command. The value of the environment variable may be changed by re-invoking the *nsevar assign* command with the shell variable name and new value. It also may be returned to the value set up before environment invocation by invoking the *nsevar assign* command with the environment variable name, but not specifying a value for it. When a child environment is created, it inherits any variables given value with the *nsevar assign* command in the parent. The NSE environment variables in the child environment may be changed by first changing them in the parent and propagating the change to the child environment with the *resync* command. It is possible to change an environment variable in a child environment with the *nsevar assign* command in interactive mode. However, once an environment variable is changed in a child environment, NSE will not permit the child environment to return changes to the parent environment via the *reconcile* command. Because an environment variable can serve as a parameter to a tool, and effect the output of the tool, the restriction on the use of the *nsevar assign* command prevents a child environment with a different variable setting from placing incompatibly derived objects in the parent environment.

In terms of parameterization management, NSE provides a developer controlled scope for the effect of different kinds of parameters by containing their impact to an activated NSE environment. It also allows for the tracking of parameter values in the Makefile as a build script by managing the Makefile in the same manner as a product source file. Finally, parameter values stored in environment variables are easily reproduced from one invocation of a tool to the next by allowing them to be registered and maintained within the context of the NSE environment, which also contains the configuration of source on which the tool is to operate.

6. Managing the Tool Application Context

When tools make use of facilities that are assumed to be available as part of the computing environment, then these facilities also should be added to configuration control to assure the proper operation of a tool. When such facilities consist of executables such as *grep* or *diff*, then NSE execset may be used to capture a version of these executables along with the executable of the tool that invokes them.

Other common system facilities that tools rely upon are UNIX system include files and system library files. Such files may not only differ from architecture to architecture, but also from one operating system version to the next. NSE execsets can be used to provide the correct versions of these files because they are not restricted to containing only tool executables. They can also be used to provide transparent access to auxiliary files necessary for system build but not managed as source files in environments. For example, if an NSE environment is setup on a Sun3 to produce code for both the Sun3 and Sun4 Computers, then the execset family attached to the environment can block out the directory containing system-specific include files such as *floatingpoint.h*. Two variants of the environment should be set up, one containing code compiled on the Sun3 for the Sun3, the second containing code compiled on the Sun3 but for the Sun4. The file *floatingpoint.h* from the Sun3 can then be installed on the Sun3 execset family member, while *floatingpoint.h* can be copied from the Sun4 and installed in the execset member that contains the tool for cross-development from Sun3 to Sun4. This will ensure the appropriate application of system provided files in the appropriate variant.

NSE execsets may contain any file which is relevant to the development context. This includes not only tool executables but also system-supplied utilities, libraries and interfaces. Providing a way to manage all these factors that affect the output of a tool can improve the stability of the application context occurring during development.

7. Conclusions

The major issues in tool version management to support software development are tool version selection, stability of tool parameterization and reproducibility of derivation context. Sun's Network Software Environment has a solution that supports tool version management in a transparent manner. Although NSE is proprietary to Sun, the mechanisms can be applied as well on any UNIX-based system.

Two important mechanisms to NSE's implementation are the per process mount of the file system and the translucent file server. These provide the transparent access and the mapping of different file and tool versions on a per process basis. The same mechanisms are used to provide source and derived object versions and tool versions.

NSE provides a context for the application of tools through the environment mechanism. It automates the selection of a tool family, and the selection of the correct object deriver from within the tool family. Environment variables are reset in the context in which the tool will operate.

The mechanisms provided by NSE can be used to solve some concrete problems in tool management: the selection of tools for development for multiple systems, the installation of incompatible tool upgrades and for organizing access to tools. However, its solution does not extend to mechanisms to manage the insertion of new tool releases that result in new versions of tool sets. Also, if tools are to be supplied to an environment, they must be present in one flat structure. There is no ability to combine or compose tool sets for use within one application context. Within the one flat structure, there is still all the potential for tools to clash in name or in parameter settings. Finally, NSE does not contribute any mechanism to enforcing the immutability of tools. Policy and conventions must be developed to govern the replacement of tools in toolsets or the creation of new toolsets.

Overall, NSE provides a good approach to serve a developer the correct versions of source and tools. It also provides mechanisms to capture some of the many kinds of tool parameters and development context.

References

[1] *Network Software Environment: Installation and Administration*
Sun Microsystems, Inc., 1988.

[2] *Network Software Environment: Reference Manual*
Sun Microsystems, Inc., 1988.

[3] Courington, William.
The Network Software Environment.
Technical Report, Sun Microsystems, Inc., 1989.

[4] Susan A. Dart, Robert Ellison, Peter H. Feiler, and A. N. Habermann.
Software Development Environments.
IEEE Computer , November, 1987.

[5] David Hendricks.
The Translucent File Service.
In *New Directions for UNIX. Proceedings of the Autumn 1988 EUUG*. European UNIX System User Group, 1988.
Dates: 3-7 Oct. 1988, Location: Cascais, Portugal.

Table of Contents

List of Figures