PRODUCT ADMINISTRATION THROUGH SABLE AND NMAKE

Steve Cichinski and Glenn S. Fowler

***Steve Cichinski** is a supervisor in the Software Systems Department of AT&T Bell Laboratories in Liberty Corner, New Jersey. Mr. Cichinski joined the company in 1977 and is product manager of SABLE. He has a B.S. and an M.S. in mathematics from Rutgers University. **Glenn S. Fowler** is a member of technical staff in the Advanced Software Department of Bell Laboratories in Murray Hill, New Jersey. Mr. Fowler joined AT&T in 1979 and is working on software development environment research, including NMAKE, automated software shipment and installation, and related algorithms. He has a B.S.E.E., an M.S.E.E., and a Ph.D., all from Virginia Polytechnic Institute, Blacksburg, Virginia.*

Product administration is a strategic technology for managing software products. AT&T has developed two software systems, *SABLE* and *NMAKE*, that support a software product throughout its life cycle and contribute to good software project management. SABLE combines the traditional software administration role of source code control and modification request tracking with support for quality assurance, document generation, interproject communications, security, and software cost analysis. Using software components managed by SABLE, NMAKE controls software product manufacture by automatically maintaining source file interdependencies to build an up-to-date product. With SABLE and NMAKE, developers can also share official versions of code while maintaining private development areas.

Introduction

As software systems have become more complex, so has the concept of *software products*. Today's software products are not merely code. They must be well-designed systems with appropriate documentation, and be easy to install, learn, use, and maintain. The Software Technology Laboratory of AT&T Bell Laboratories has reorganized its software development process to meet these new, higher standards.

In this article, *product* refers to customer deliverables as well as internal or intermediate items, such as requirements documents and test scripts needed to produce deliverables. *Deliverables* may include software, hardware, firmware, documentation, or a mixture of these. *Project* refers to the staff and technical/administrative subculture responsible for creating the product. *System* refers to the software portion of the product.

Older software administration systems concentrate on source control and/or modification request tracking. With source control, users can track precisely what has changed from one version of the source to the next and restore old versions of the source using information about what has changed. By tracking *modification requests* (i.e., MRs, requests for product enhancements or fixes), users can track the progress of requests for change.

The SABLE product administration system developed by AT&T has expanded the traditional software administration role of source control and modification request tracking by both providing more capability in these areas and branching out into related areas such as quality assurance, document generation, inter- and intraproject communications, security, software manufacture, and cost of rework. SABLE users benefit from the added control over the development process and over intraproject communications. With SABLE, it is clear who is working on what, what state it is in, and what has been changed for each release. SABLE is in use or being evaluated by more than 180 projects throughout AT&T.

The initial requirements for SABLE were defined by an expert software administration team. The team examined many existing systems and chose the Change Management System as a suitable baseline from which to build the first release of SABLE. Since then, there have been two additional releases that have included enhancements requested by customers from AT&T as well as concepts from older systems.

Beyond the need to maintain a complete, recoverable product component history is the need to build a usable product. In SABLE, *build* control is handled by a software manufacturing tool known as NMAKE.[1] NMAKE is efficient, supports concise build specifications, allows common specifications to be shared within and between projects, and is especially well-suited to the SABLE management paradigm. NMAKE is used in more than 250 AT&T projects.

NMAKE is an evolution of the UNIX® system build programs `make`,[2] `augmented-make`,[3] `build`,[4] and `mk`.[5] It has been designed to handle the dynamic nature of software development, especially where many developers are working on a single software product. There is also a consistent, maintainable partitioning of information in NMAKE. This partitioning allows various groups of users to share common actions, while also providing per-user overrides for these actions.

The combination of SABLE and NMAKE provides extensive product administration support. SABLE administers product *source* files by maintaining a complete history of modification requests and the resulting source file changes. This history allows a directory structure to be populated with specific source file versions from any point in the product lifetime. NMAKE administers product *target* files by using the source files in a SABLE-populated directory structure and its own history information to generate up-to-date product targets.

SABLE and the Product Life Cycle

SABLE is a support system for the product development process. As such, it is important that SABLE fit the structure of a well-run process yet be flexible enough for local variations. The following sections examine the typical phases in a product's life cycle and describe the support SABLE provides for each.

Product Documentation. A document is usually the first tangible output in the life cycle of a product. It may be a business case, a feasibility study, a research finding, or a joint working agreement. In whatever form, it is an important historical document that should be preserved and available to project members throughout the product's life. In many projects, several other equally important documents, such as a project plan[6] or a quality plan,[7] are produced soon after this initial document. Detailed specifications and documents are continuously added and updated throughout the product's life.

SABLE supports this documentation process in several ways. SABLE has a template capability so that frequently written documents can use a standard format following organization guidelines. Several standard templates (such as outlines for annotated requirement specifications or design specifications) are supplied with SABLE. An "include" capability allows fragments of a document (e.g., a graph or a table) to be reused in other documents.

A "collection" capability allows a group of related documents to be treated as a single entity. The user can print a complete copy of a user guide by specifying the name of the collection rather than a complete list of the files that make up the guide. NMAKE is used behind the scenes, without user involvement, to print the up-to-date version of the collection. Similarly, a global change can be made to the guide by specifying the change and the name of the collection.

Documents can be saved so that changes to the document source can be tracked, as is done for source. Modification requests (MRs) can be entered and tracked against documents. There are even MR states (e.g., inspect and publish) that are specifically for documents. Quality metrics for documents can be calculated and sent to the AT&T Bell Laboratories Quality Assurance Center for inclusion in quality assurance reports.

Product Development. Every time a product is modified, it is modified for a reason. In SABLE, these reasons are captured in MRs. Whenever any source under SABLE is changed, the change and the reason for the change are recorded and an association is established between the MR and the related source change. With this information, a user can always tell how and why the product was changed and can recreate specific versions of the product.

As work progresses in response to the modification request, the MR passes through a series of states on its way to resolution. During this time, each person or team responsible for the next action to be taken is automatically notified of the transition. In this way, SABLE can pace the development process. MRs are created, accepted, assigned, submitted for test, integration-tested, system-tested, approved, and closed. (This is only a subset of available states. Other states handle situations that may arise in the life of an MR, thus giving project members the flexibility to manage the development process on a day-to-day basis.)

Software projects without a product administration system often waste considerable time when they must freeze a source version to do a system build and monitor changes made by developers who are not working directly on the build. Build problems usually arise when the source does not integrate. In trying to get the system built, developers make changes to their version (so that it will build), while at the same time, other developers are changing local source copies for the next build. Because these local copies don't reflect the build team changes, the next build is delayed further while changes are being integrated. When a product administration system such as SABLE is used on larger projects that do regular system builds, it can track and integrate (or keep separate) changes made by the build team and the other developers. This is considerably more productive.

Product Building. Product building occurs throughout the product life cycle. Source is continually changed to test new features, to fix bugs (malfunctions), or to restore capabilities from earlier phases in the life cycle. After source has been changed, the product must be rebuilt to reflect these changes. To conserve machine resources and reduce build time, only those portions of the product affected by source changes should be rebuilt. Rather than manually rebuilding components (a potentially error-prone process), building is done automatically by a separate, specialized program. This program is described in detail in the next major section.

Product Testing. Quality has become increasingly important in the competitive marketplace. One component of a successful quality assurance program is effective, efficient testing. SABLE has a mechanism for establishing and recording the membership of test teams and uses this information when processing MRs in the test states. In addition, test teams receive a description of the features to be tested along with the MR; thus, they can save time by focusing on those features that are ready to be tested. The MR description can also give the test teams some additional information about what to look for while testing.

SABLE also provides several optional test states, such as "integration tested" and "system tested." If a test team uncovers problems, it can pass its findings back to the assigned developer in a rejection description that is appended to the original MR. SABLE automatically notifies the developer, who can use this description as a basis for investigation and resolution. This decreases turnaround time by providing faster, more complete communications.

If test scripts are used, as in automated regression testing, they can be stored under SABLE and the appropriate script versions can be retrieved via SABLE's `getversion` command (in this case, "get the specific version of the test scripts that satisfies a given set of MRs"). These MRs may describe changes needed to test the next version of the system.

SABLE accepts MRs from external systems, so that if a mechanized test system is in place, the problems found during testing can be automatically cast as MRs and shipped to SABLE. When the MRs reach SABLE, they may be entered automatically into the database or held for review and then entered selectively. This connection between a test system and SABLE eliminates manual entry and can increase the thoroughness of problem descriptions supplied with the MR.

Product Maintenance. A problem frequently encountered in maintenance is the need to reproduce the version of a product related to individual customer trouble reports. On occasion, the source for the customer's product version has been lost, either because the older version simply was not saved or because information describing the exact customer version was not recorded. Even if the problem has been identified and a solution is known, a fix can not be made because the original source version can not be located. Because SABLE, with NMAKE's help, gives users the ability to recreate prior product versions (from a list of MRs), projects are better able to solve customer problems.

Product Shutdown. There are many reasons why an organization may stop work on a product. The product may have reached the end of its useful life; it may have been turned over to the customer or another organization, as frequently happens with government and foreign contracts; or, it may have been unsuccessful for business, management, or technical reasons.

Having a record of the product source throughout the life cycle can aid the transition to shutdown. In each case, a wrap-up analysis should be done as part of a continuous improvement program. The day-to-day quality and productivity data that SABLE accumulates can be invaluable in this analysis. For example, SABLE stores every MR in its inactive MR database. When a product has outlived its usefulness, the organization may be developing a product sequel or a product similar to the previous one. Having all product-related documentation and source under SABLE makes it much easier to *reuse* designs, specifications, plans, and source code, even if they are only used as examples for future work.

The knowledge transfer that should take place during product shutdown frequently takes much longer than it should. It takes time to gather the relevant, up-to-date information. And frequently it takes even longer for those receiving this information to digest it because their knowledge of and access to the product history has naturally been more limited. Because SABLE is an orderly system and can store (historical) documents and MR history, it is easier for both sides in the transition.

NMAKE and Product Building

Software product building has traditionally been done by the `make` program. There are many variations (`augmented-make`, `build`, `mk`, and `nmake`), but all work in a similar way. A description of source file interdependencies is placed in a `Makefile`. Each of these dependencies asserts a relationship between a target file and its prerequisite files and provides an action (shell script) that builds the target file from its prerequisites. Assertions take the form:

$$\begin{array}{l} \text{target} : \text{prerequisite}_1 \ldots \text{prerequisite}_n \\ \qquad\quad \text{action} \end{array}$$

If the target file does not exist, or if its modification time is older than the modification time of any of its prerequisites, the action is executed to build (or rebuild) the target file.

Although NMAKE is similar to other UNIX system build programs, it has many additional features that make it the clear choice for build control in SABLE. The significant differences are discussed in the following sections.

Specification. Input specifications to NMAKE are partitioned into three general areas.

- `Base rules` are shared by all NMAKE users and are automatically defined for each NMAKE installation. These rules describe the local system interface and provide definitions for common actions such as:
  - `lint`: Pass all C source files through the `lint` processor
  - `cpio`: Copy all product source files to a single `cpio` archive suitable for shipping to other systems
  - `clobber`: Remove all generated targets, leaving the original source files intact.

- Global `Makefiles` are typically shared by members of a single project. These rules may describe specialized compiler interfaces and project-specific common actions.
- `Makefiles` provide source component interdependencies and build actions for each particular product. A single project may have many `Makefiles`, each describing a subcomponent of the total product.

Automatic Dependency Analysis. A common problem with any build program occurs when changes made to the source files are not reflected in the `Makefile` specification. This usually involves implicit file dependencies (often found in `include` statements) that are added to, or deleted from, source files. `Include` file modifications may go undetected if such dependencies are omitted from the `Makefiles`. Such omissions could result in insidious bugs in which portions of a single program are compiled with different versions of the same `include` file.

NMAKE solves this problem by automatically scanning input source files for `include` statements and adding the included files onto the prerequisite list of the corresponding target files. In this scheme, a source file is rescanned whenever its modification time changes, ensuring that the implicit prerequisites are always up-to-date. `Include` styles for the C, Fortran (formula translator), Ratfor (rational Fortran), and M4 (a macro processor used as a front end to Ratfor) languages are directly supported for efficiency. New languages are handled by supplying an external scanning program that is called by NMAKE for each source file.

State Database. NMAKE is the only UNIX system build program that maintains *state* information in its own database. Other build programs rely solely on file modification times automatically maintained by the UNIX system. Without the state database, a target file is out of date when its modification time is older than the modification times of any of its prerequisite files. The state database allows NMAKE to:

- Retain automatic `include` file dependency information, recomputing only when individual source files change.
- Do accurate file-modification time comparisons. A file is out of date if its current modification time is different from the database modification time.
- Allow "state variable" dependencies by retaining NMAKE variable values and noting changes between the current and database values.
- Note that a target is out of date if its prerequisite list changes.
- Note that a target is out of date if its action changes.

Viewpathing. In most projects, source changes are made concurrently by a group of developers. As mentioned above, an official source change requires the cooperation of all developers to ensure proper integration. *Viewpathing* is a technique that allows each developer to share code while maintaining private development areas. A *viewpath* is an ordered sequence of directory hierarchies in which files occurring earlier in the sequence override files occurring later. In practice, the last view (lowest precedence) contains a complete copy of the official source. With viewpathing, each developer need only copy into the private view that portion of the product that is different from the official view. Figures 1a and 1b show the physical directory hierarchies for both private (shown in gray) and official (shown in gold) source. Figure 1c shows the logical view provided by overlaying the private view on top of the official view. Not only does viewpathing avoid the problem of each developer having a complete copy of the official source, but each developer's physical view clearly shows the official files that have been changed. A viewpath may be many levels deep, with each view representing a different phase of product development.

Although viewpathing is a simple concept, it poses some implementation difficulties. Build provides top view file names for viewpathed files by temporarily creating hard links to files in lower views. A minor drawback is that a viewpathed file must be copied to the top view (rather than linked) if it resides on another file system. More serious problems arise when two or more builds run simultaneously. One build can remove temporary links before the others are finished operating on them. Finally, system crashes may result in temporary links suddenly becoming permanent, effectively moving lower view files to the top view. It is very difficult to recover from this last class of errors.

An operating system or shared library implemen-

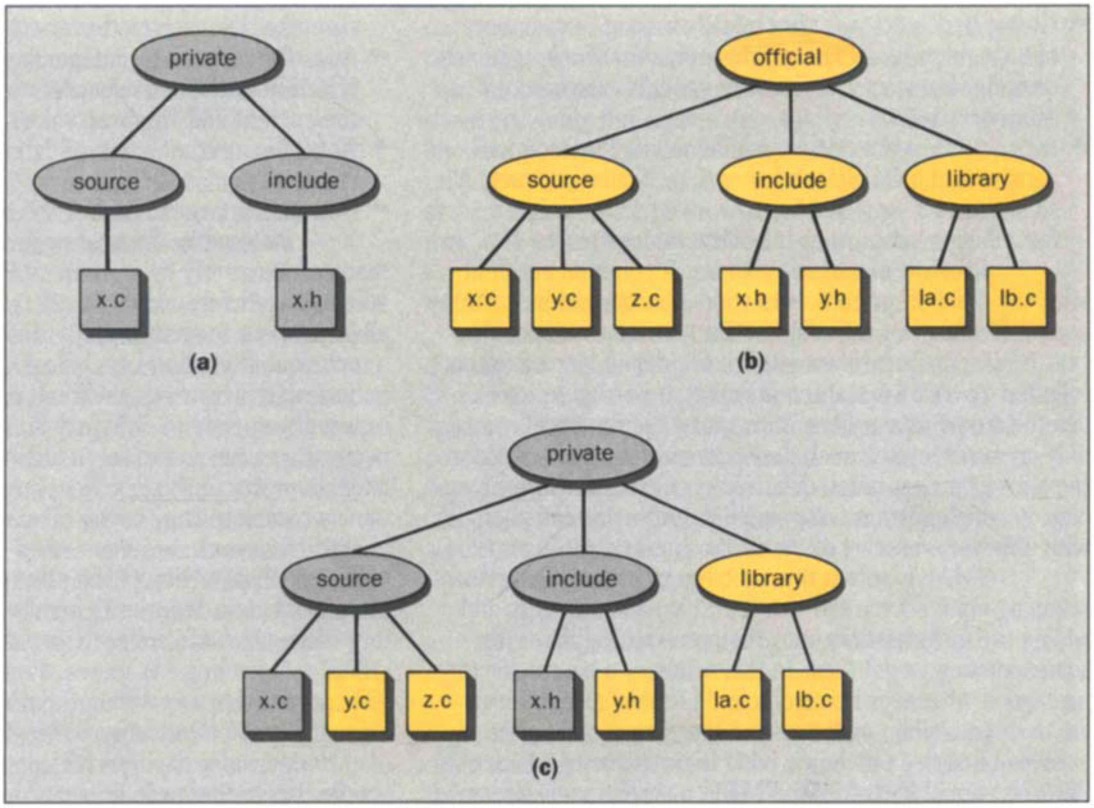


Figure 1. The viewpathing feature in NMAKE allows developers to test alternate build strategies of a product using only their own changes against a stable background. (a) Private view; (b) official view; and (c) private view imposed on the official view.

tation is ideal because every program can have viewpathing without change.[8] This requires operating system changes, however, because viewpathing is not a standard UNIX system feature. These changes are not easy to distribute, especially in light of the many UNIX system implementations on the market. (AT&T does not rely on any single UNIX system implementation.) The alternative approach, and one that requires no standard UNIX system program changes, is to place viewpath capability in a few specialized programs that pass physical rather than logical file names to other system programs. This is the mechanism currently used by NMAKE. A research version that uses operating system viewpathing is planned.

Execution. NMAKE has an efficient interface with the UNIX system shell that saves time when many actions are executed for a single build. Only one shell is executed for an entire NMAKE run. In addition, each action is sent as a complete unit, so that no restrictions are placed on the action content. Other build programs either send each action line or, at best, each action to a new shell.

The NMAKE/shell interface also supports concurrent action execution (as does `mk`). The user simply specifies the maximum number of concurrent actions. Concurrency is transparent and requires no `Makefile` changes. *Semaphores* (signals) are provided for the rare cases in which mutual exclusion is required within a group of related actions.

Instead of being executed by the shell, actions may be sent back to NMAKE to be read as `Makefile` input. This mechanism is the key to the expressive power of `base rules` and global `Makefiles`.

Results. NMAKE has contributed to many projects within AT&T, including some projects that have not used SABLE. Users have reported various improvements with NMAKE including:

- Single processor build time decreased by a factor of two.
- n-processor build time decreased by nearly a factor of n (depending on multiprocessor implementation).
- The size of an input `Makefile` decreased by a factor of 8 to 20.
- The number of input `Makefiles` per project could at times be reduced to one.
- The build time when all targets are up-to-date decreased by a factor of 4.

SABLE and Project Management

The SABLE architecture is shown in Panel 1. SABLE was designed to support efficient development in production shops. As such, it is based on the simple principles of order, shared knowledge, clear responsibility, effective communication, subdivision of large tasks, and built-in quality. SABLE gives project managers wide latitude in determining the degree to which these principles are applied in each project. SABLE is a tool that supports good project management, but it is *not* a substitute for it.

Organizational Efficiency. For an organization to function efficiently, large tasks must be broken down and responsibilities assigned clearly. Priorities and due dates must be set. SABLE supports organizational efficiency in several ways.

When an MR is accepted, it is assigned to a person or a team. A priority and due date may also be set. The persons assigned to the task are notified. If necessary, an MR may be reassigned or even unassigned. If the MR task is large or spans several areas of responsibility, it may be subdivided into other MRs (called *spawned* MRs); these may each be assigned to the appropriate staff members. For example, a new feature in a software product may require changing an application program, the human/machine interface, the database, and the user guide. Each of these areas may be the responsibility of a different team. With spawned MRs, this task can be broken down into smaller tasks that can be assigned, tracked, and managed separately.

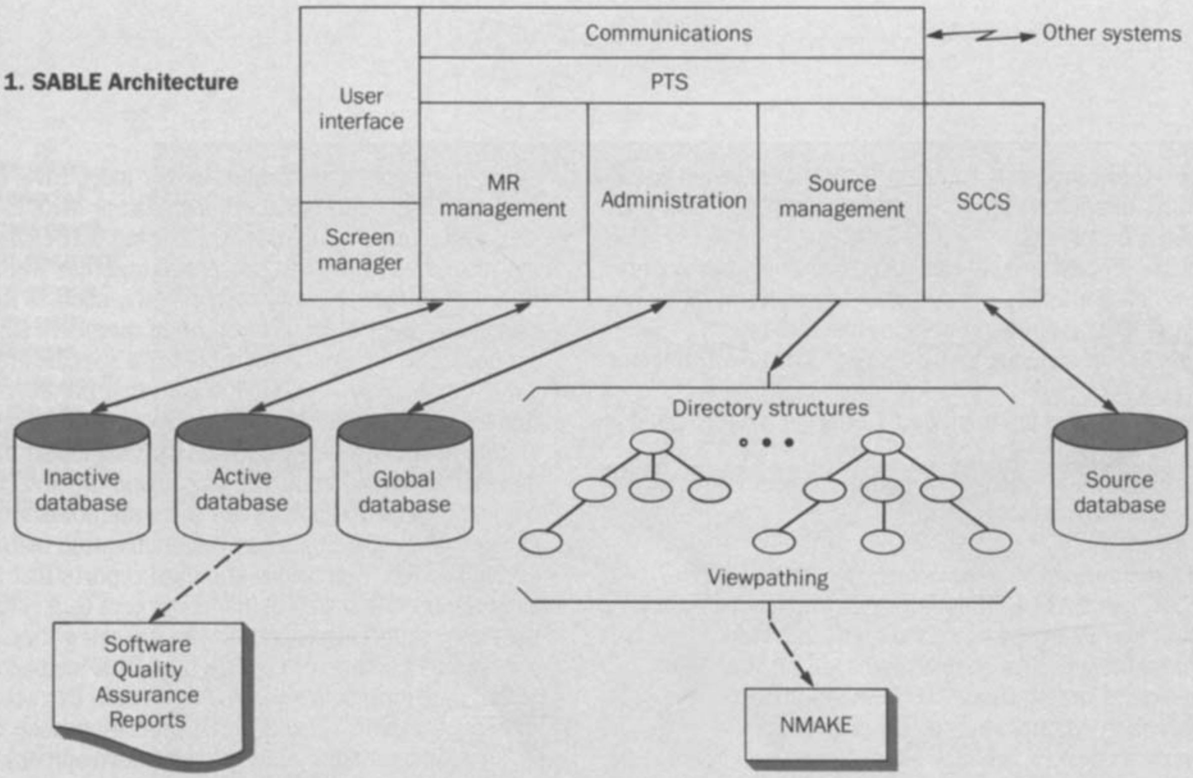
If tasks described in two or more MRs turn out to be interdependent, SABLE allows these MRs to be declared mutually dependent, assuring that changes resulting from these MRs are processed together when necessary. Users can also group MRs, such as for an upcoming release. In addition, staff members can be grouped into a team responsible for a specific area.

Communication. One reason that projects run into difficulty is because there is a lack of communication among team members. Both SABLE's support for documentation (formal communication) and the automatic communication to appropriate team members when an MR changes state are important communication features. In addition, SABLE provides standard reports that are typically required in a well-managed project (e.g., MR reports, summary reports grouped by various categories, or changes to a source file by MR). SABLE also allows users to customize reports and provides a data extract file so that users can devise their own special-purpose reports.

In larger projects, product development may be done on one machine and system test on another. SABLE operates in a multimachine environment (host and satellite) in which the local area network can be either a Datakit™ network multiplexer or an Ethernet™ network. (Ethernet is a trademark of Xerox Corporation.) It also works with UNIX System V, Release 3 remote file sharing, which allows files on one machine to be accessed from another machine.

Different organizations often cooperate on developing a product. For example, a systems engineering organization and a development organization might work together on a single product, or teams from two existing products might work together to support an interface between them. It is important for these different organizations to share MR information, both MR descriptions and MR status. A development organization might discover that a change is needed to both the software and requirements; the MR would then be forwarded to the systems engineering organization. Or, a factory might encounter problems with a hardware design, possibly affecting other product-related organizations. Because MR information can be sent from one SABLE instance to another, SABLE can support these types of situations.

Panel 1. SABLE Architecture



User Interface/Screen Manager. Provides full-screen and command-line interface. Table-driven techniques are used; users may make certain project-specific changes to the interface.

Communications. Handles a SABLE instance spanning several machines, SABLE-to-SABLE communication, and communications with other systems such as a regression test system or other MR sources.

PTS. The personnel tracking system handles SABLE logins and administrative and profile data for each user.

MR Management. Provides the modification request tracking and reporting function.

Administration. Gives a project the control needed to establish SABLE and tune it for the project environment.

Source Management. Provides the source change tracking and control function. The source code control system[9] (SCCS) is used to provide part of this functionality.

Databases. The *active database* stores information about open MRs and the source changed as a result. The *inactive database* is identical to the active database and stores information about MRs that have been closed. These data are useful for project wrap-up analysis. Separating open and closed MRs speeds access to the open MRs. The *global database* holds information about the different products stored in SABLE. Project customization features are supported by the data stored here. SCCS manages and stores the product in the *source database.*

Directory Structures. SABLE populates these via its `getversion` command. One directory structure shown here could be partially populated and used by a developer for recent changes. The other, fully populated, could be the official source directory structure. NMAKE's viewpathing capability is used to build a version of the product incorporating the recent changes from the developer's directories and the rest of the system from the official directories.

Software Quality Assurance Reports. The data for these reports are collected from SABLE's databases and formatted to produce reports locally or to transmit to the AT&T Bell Laboratories Quality Assurance Center.

Quality Assurance. SABLE, through its support of the product testing phase discussed above, is by nature a quality assurance tool. Because SABLE keeps records of MRs as they travel through the system, it can supply a quality assurance team with a great deal of status and history information. In particular, it provides data for the quality metrics distributed by the AT&T Bell Laboratories Quality Assurance Center. SABLE also provides an extract file that can be used to produce customized quality assurance reports.

Access Control. For software, firmware, and documentation projects, source *is* the product and must be protected against unwanted or unauthorized changes. To this end, SABLE provides several access control mechanisms. To execute SABLE commands, a user must have a SABLE login. The SABLE administrator is the only user who can add logins, which are stored in SABLE's personnel tracking system.

For a user to change source under SABLE, not only must a valid MR identification be given, but the MR also must be assigned to the user. This helps assure that source is modified for the right reasons and by the assigned staff.

Projects have the option of recording the history of all SABLE activities. This history serves as an audit trail and tracks when changes occurred and who made them. These mechanisms form a second line of defense and, obviously, depend on appropriate security precautions being in place for the equipment on which SABLE runs.

User Considerations

SABLE has many features that make it easy to use and customize.

The Human/Machine Interface. SABLE provides both a full-screen and a command-line interface. The full-screen interface runs on standard terminals used at AT&T and has pop-up menus, as well as help and error messages. Figure 2 shows a typical SABLE screen used to create an MR.

Help messages pop up on demand and error messages pop up as errors occur. Once a menu has appeared, the system uses a first-match algorithm to select a choice based on what the user has entered. This speeds data entry. The interface provides standard field-navigation capabilities as well as some other, more advanced capabilities, including multiple selections from a pop-up menu. For fields that accept lists, where the maximum allowable list size is much longer than the usual list size, there is a left-right (horizontal) scrolling capability so that all necessary data can be entered.

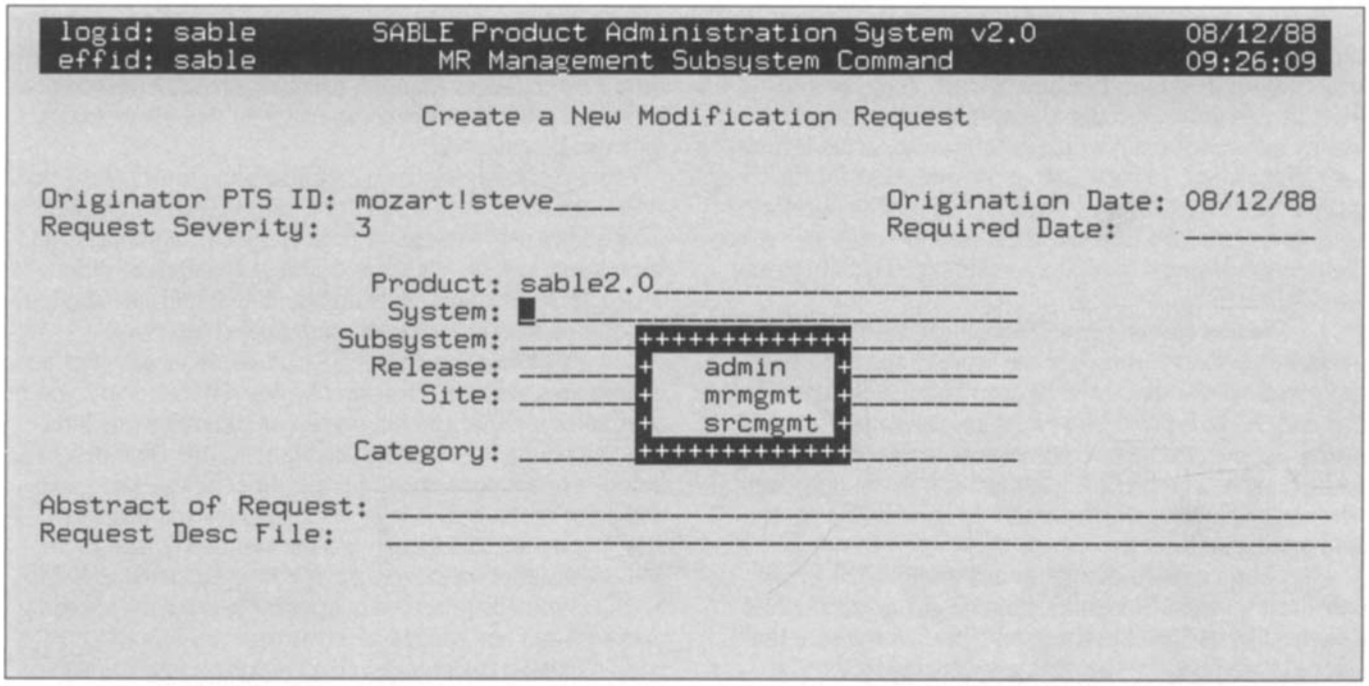
SABLE provides a command-line interface so that users can write shell scripts that call SABLE commands. This allows users to create project-specific commands and report generators. Where not limited by inherent differences in the terminal, the command-line interface and the full-screen interface exhibit comparable behavior.

User Customization. A SABLE user can set interface choices in a personnel tracking system (PTS) record. Each user selects either the full-screen or command-line interface, an editor for entering descriptions, the time delay before a menu pops up, and either brief or extended help and error messages. A user also specifies administrative information such as home machine and phone, and room and organization numbers. After a user has been added to SABLE, immediate access is granted to the corresponding data so it can be changed at any time.

Project Customization. Because each project may have different needs, either for historical, managerial, or contractual reasons, SABLE provides flexibility at the project level as well. There are some project-wide choices for those fields that will appear on the screen. These choices are based on whether the fields are mandatory, whether project-specific templates will be used, whether a complete history will be kept, and what test states and user execution permissions for specific command sets will be used. Because the valid values that appear in many of the pop-up menus are project-specific, these values may be set according to a project's needs. The project can set up a menu tree so that a value selected for one field may restrict values for later fields. (In Figure 2, the menu tree has been set up so that only valid values appear on the *System* menu.) Because SABLE runs on all UNIX system machines in common use throughout AT&T, projects also have considerable flexibility in choosing the machine on which to run SABLE.

Future Directions

SABLE is already a robust product administration system by today's standards. But because of its place in



```
 logid: sable        SABLE Product Administration System v2.0             08/12/88
 effid: sable              MR Management Subsystem Command                09:26:09

                         Create a New Modification Request

Originator PTS ID: mozart!steve____                    Origination Date: 08/12/88
Request Severity:  3                                   Required Date:    ________

                    Product: sable2.0____________________
                     System: █___________________________
                  Subsystem: ______+++++++++++++_________
                    Release: ______+   admin   +_________
                       Site: ______+   mrmgmt  +_________
                                   +  srcmgmt  +
                   Category: ______+++++++++++++_________

Abstract of Request: _________________________________________________________
Request Desc File:   _________________________________________________________
```

Figure 2. A SABLE pop-up menu with project-specified values. In this example, the user waits to see the valid values for the *System* field and the menu for that field then appears. Once a project-specified value has been selected for *Product*, only certain values are valid for *System*.

the development process and the ever-increasing demands placed on the AT&T development community, there is an ongoing research and development effort for both SABLE and NMAKE. Below we discuss some possible future extensions and suggest potential benefits in the product administration area. Many of these features are planned for future releases; others depend on measuring the possible benefits to customers.

Controlling Non-ASCII Text. At present, the standard source control system used in SABLE can store only ASCII (American Standard Code for Information Interchange) text. In some projects, it would be beneficial to control non-ASCII text. For example, there is a need to control object code as well as source code when doing incremental compiles, as well as a need to control digitized data for pictures. A future release of SABLE will provide this capability, along with changes in size limitations and improved speed of the source control system.

Software Manufacturing. SABLE and NMAKE are powerful, independent tools in their own right. Because of this independence, however, the user is sometimes unnecessarily exposed to SABLE-NMAKE interface details.

Currently, each time a `getversion` command is performed, SABLE does not know if a version of the product will actually be built from that source or if the source will be modified outside SABLE before the system is built. Similarly, NMAKE does not know whether the source it is working with came from SABLE. The result is that, despite good intentions, one can never be sure, without additional checking, if what left SABLE was actually used to build a product because of the uncontrolled window between SABLE and NMAKE.

Our goal is to provide better support for software manufacturing by asserting more control over the SABLE-

NMAKE handshake. Then, not only will there be certainty about what is being built, but we will automatically be able to keep complete records about when the system was built, what went into the system and who requested the build.

Documentation. The documentation feature provided by SABLE will continue to expand. Because SABLE can be the document repository for a project, we want to add document classification and search capabilities so that documents can be found quickly, giving new team members easy access to relevant documentation. Authoring tools and electronic document ordering and distribution are other possibilities.

More Support for Hardware Projects. SABLE has its roots in software administration, but its goal is to support hardware projects equally well. SABLE tracks hardware MRs as well as it tracks software MRs. We have plans to support hardware quality metrics just as we support software and documentation quality metrics. The document templating capability works as well for hardware-specific documents as any others and we will investigate the standard hardware-specific document templates that should be provided with SABLE. The ability to control non-ASCII text, discussed above, may be of value in storing digitized data and schematics.

Rework Cost and In-Process Metrics. Because SABLE collects additional information each time an MR changes state, it can gather data on actual-versus-scheduled work estimates. Analyzing the reasons for change and the amount of rework required can give a manager additional insight for development process improvements. Similarly, the collection of in-process metrics, like inspection results, can help a manager make midcourse corrections. Because SABLE already keeps MR data, it would be natural to collect in-process quality metric data or, at least, cooperate with a system that does. We will be investigating customer needs and options in this area with the goal of providing an integrated tool environment.

Source Analysis. There is a body of useful tools that supports what might be called "source analysis." Some examples are software module complexity metrics (see McCabe[10] and Halstead[11]), software structural complexity metrics and the related software structure graphs (`dag`,[8] a directed graph drawing program), document analysis, software information abstractors that support a data dictionary population (`cia`,[8] the C information abstractor), and consistency and standards checkers. SABLE can provide and/or support these types of analyses because it stores product source.

Because of SABLE's ability to recreate different versions of source for analysis, A-B comparisons can be easily done. For example, project managers can see the effect of a new practice or technology they've put into place. In addition, because SABLE maintains MR data, a project should also be able to address those characteristics that affect its bug rate.

SABLE can act as a source gatekeeper, and, *if necessary*, can run selected analyses as source is stored. SABLE can also support other analyses such as comparing requirements document characteristics with the resulting software, or comparing software to be tested with the corresponding test scripts. Because SABLE tracks source changes, incremental analysis data updates would also give a better indication of changes since the last analysis was done. NMAKE with, perhaps, additional `base rules` would be the natural engine to run these analyses.

Summary

The SABLE/NMAKE combination has made significant and tangible contributions to software product administration at AT&T. Each program, although useful on its own, complements the other to provide comprehensive support for a software product throughout its life cycle. Through active research and development, we hope to improve on the product administration benefits already seen.

Acknowledgments

Jonathan Shopiro and Mark Freeland worked on the initial NMAKE viewpath design. Dave Korn, Eduardo Krell, and Jerry Schwartz tried out NMAKE, and John Sobek has made major contributions to the SABLE development effort.

References

1. G. S. Fowler, "The Fourth Generation Make," *Proceedings, USENIX Portland 1985 Summer Conference*, pp. 159-174.

2. S. I. Feldman, "Make—A Program for Maintaining Computer Programs," *Software—Practice and Experience*, Vol. 9, No. 4, April 1979, pp. 256-265.
3. "Augmented Version of Make," *UNIX System V—Release 2.0 Support Tools Guide*, April 1984, pp. 3.1-3.19.
4. V. B. Erickson and J. F. Pellegrin, "Build—A Software Construction Tool," *AT&T Bell Laboratories Technical Journal*, Vol. 63, No. 6, Part 2, July-August 1984, pp. 1049-1059.
5. A. G. Hume, "Mk: a successor to make," *Proceedings, USENIX Phoenix 1987 Summer Conference*, pp. 445-457.
6. L. B. Robertson and G. A. Secor, "Effective Management of Software Development," *AT&T Technical Journal*, Vol. 65, No. 2, March/April 1986, pp. 94-101.
7. *Quality by Design*, Issue 1, AT&T Bell Laboratories Quality Assurance Center, 1986.
8. D. G. Belanger, G. D. Bergland, and M. Wish, "Some Research Directions for Large-Scale Software Development," *AT&T Technical Journal*, Vol. 67, No. 4, July/August 1988, pp. 77-92.
9. *UNIX System V, Release 3 Programmer's Reference Manual*, AT&T Customer Information Center, Indianapolis, Indiana, 1986.
10. T. J. McCabe, "A Complexity Measure," *IEEE Transactions On Software Engineering*, Volume SE-2, Number 4, December 1976, pp. 308-320.
11. M. H. Halstead, *Elements Of Software Science*, Elsevier, North Holland, 1976.

(Manuscript received April 26, 1988)