

SABLE/NMAKE - Technology for Product Administration

*S. Cichinski
Glenn. S. Fowler*

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

1. Introduction

Product Administration is a strategic technology for managing a product during its life cycle. To stay competitive, we have to be able to build and deliver higher quality products faster and we have to be able to do it throughout the life of the product, not just for the first release. Not only are the products we are offering becoming more complex, but our concept of what a product is is broadening too. The product is no longer just code. It is a well-designed system, with appropriate documentation, that is easy to install, learn, use and maintain and that meets the customers' quality and service needs as well as their technical needs. We have been reorganizing our development process[1] so that we can build products that meet these new, higher standards.

Similarly, **SABLE** is expanding the traditional Software Administration role of source code control and modification request tracking by both providing more capability in these areas and branching out into other related areas such as quality assurance, document generation, inter/intra-project communications, security and cost of quality. Users throughout AT&T have been benefiting from added control over their development process and the focused intra-project communications that **SABLE** affords. It is clear who is working on what, what state it is in and what has been changed for an upcoming release.

Major components of a software product are source files and target files. Source files are the basic building blocks from which the target files are generated. A deliverable software product is a collection of target files that have been generated from a much larger collection of source (code, documentation, diagrams) files. **SABLE** administers product source files by maintaining a complete history of modification requests and the resultant source file changes. This history allows a directory hierarchy 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 the **SABLE** directory hierarchy and its own history information to generate the up-to-date product targets.

NMAKE is an expert product generation tool. It automatically detects source and target interdependencies and uses them to propagate source file changes to the generated target files. Several **NMAKE** features make it an attractive companion to **SABLE**:

- Concise target generation action specifications,
- Default actions are easily adapted to the special needs of each product,
- *viewpathing* allows source files to be partitioned into official and developer-specific directory hierarchies,
- The state data base reliably detects when source files and prerequisites have changed,

AT&T – PROPRIETARY
Use pursuant to Company Instructions.

- The **NMAKE** language simplifies and reduces user input,
- Automatic support for concurrent generation of target files.

Both **SABLE** and **NMAKE** are fully supported and have active research and development programs.

2. **SABLE** and the Product Life Cycle

SABLE may be thought of as an operations support system for the product development process. As such, it is important that **SABLE** help structure and fill the needs of a well-run process, as well as being flexible enough to support necessary local variations. Below we look at typical phases in a product's life cycle and describe the type of support **SABLE** provides in each.

2.1 **Product Documentation Phases**

A document is usually the first tangible output in a product's life cycle. It may be a business case, a feasibility study, high-level specifications, a research finding or a joint working agreement, but whatever it is, it is an important historical document that should be baselined, updated as time goes on and be available to product team members throughout the product's life. Well-run product teams quickly follow this initial document with several other, equally important documents, such as a Project Plan[2], a Quality Plan[3], more detailed specifications and documents are continuously added and updated throughout the product's life. Indeed, one of the requirements of a good development methodology[1] is to formalize communication across organizations and across time.

SABLE supports the generation and use of documentation in several ways:

- a. The template capability makes it easy to reuse standard formats for frequently written types of documents and to follow organization standards. Several standard templates are supplied with **SABLE**,
- b. The collection capability allows a group of related documents (e.g. chapters of a book or sections of a Project Plan) to be treated as a single entity, when it is convenient for the user to do so,
- c. The include capability allows portions of a document (e.g. a graph or a table) to be reused in other documents.
- d. Documents can be baselined and changes to document source can be tracked just as is done for software source,
- e. Modification Requests (MRs) can be entered and tracked against documents. In fact, there are MR states (e.g. inspect and publish) that are specifically for documents,
- f. Quality metrics for documents can be calculated and sent, when desired, to the ATT-BL Quality Assurance Center for inclusion in the semi-annual Software Quality Assurance Reports (SQAR).

2.2 **Product Development Phase**

Every time the product^{*} is modified, it is done so for a reason. In **SABLE**, these reasons are captured in something called Modification Requests (MRs). Whenever any source under **SABLE** is changed, the reason for the change, i.e. the MR Id, is recorded and an association is established between the MR and the source changes for that MR. This serves two purposes, the user:

* A product could be hardware, software, firmware, documentation or a mixture thereof.

- a. Can always tell why and how the product was changed,
- b. Can always recreate specific versions of the product.

As work in response to the MR progresses, the MR passes through a series of states on its way to resolution. When the MR changes state, the person or team responsible for the next action to be taken in response to the MR is automatically notified of the transition. In this way, **SABLE** acts as a pacemaker for the development process. MRs are created, accepted, assigned, submitted for test, integration tested, system tested, approved and closed⁺.

Software projects that don't use a product administration system often waste considerable time when they have to both freeze a version of their source so they can do a system build and keep up with changes being made a majority of their programmers, who are not working directly on the build. What usually happens is that there are build problems because the source didn't integrate. The programmers who are trying to get the system built are making changes to their version so it will build and are not allowing any other changes because these might cause new integration problems. Meanwhile, the rest of the programmers are working on the changes for the next build in their local copies of the source. But these local copies don't reflect the changes the build team made, so when it is time for the next build there is another delay while the changes are integrated and so on. On larger projects that do regular system builds, say, for system test, productivity increases considerably when a product administration system like **SABLE** is used because it can track and integrate or keep separate, as needed, changes made by the build team and the other programmers.

2.3 Product Testing Phase

Quality has become increasingly important in the competitive marketplace. One component of a successful product quality assurance program is one or more effective, efficient test teams. **SABLE** supports testing in the following ways, it:

- a. Provides 5 test states (all optional), such as integration tested and system tested. This allows the product team to tailor an appropriate testing program for their needs.
- b. Records the establishment and membership of test teams formed to carry out the testing program and uses this information when processing MRs in the test states. chosen testing phases,
- c. Makes the test teams more efficient because the features to be tested are described in the MRs that are delivered for test. This saves the test teams from wasting time testing features that are not ready for test and focuses their attention on the features that should be tested. The MR description can also give the test teams some additional information about what to look for while testing.
- d. If test scripts are used, as in automated regression testing, they can be stored under **SABLE** and the appropriate versions of the scripts can be pulled out and used for the specific testing required.
- e. **SABLE** has the ability to accept MRs from external systems, so if there is a mechanized test system in place, the problems it finds during testing can be automatically cast as MRs and shipped to **SABLE**. When the MRs reach **SABLE**, they may be automatically entered into the data base or they may be held for review and selectively entered. This connection between a test system and **SABLE** eliminates the

⁺ This is a subset of the available states. Others are provided to handle situations that may arise in the life of an MR, thus giving product team members the flexibility needed to manage the development process on a day-to-day basis.

manual entry effort and can increase the thoroughness of the problem description supplied with the MR.

2.4 Product Maintenance Phase

One of the frequently encountered problems in maintenance is being able to recreate the version of the product with which the customer is having trouble. On occasion, the source needed to build the customer's version of the system has been "lost" and even if the reason for the problem has been identified and the solution is known, the customer cannot be sent a fix because there is no source to fix. Since **SABLE**, with **NMAKE**'s help, gives users the ability to recreate prior versions of the product, they are not prevented, because of lack of source, from recreating and fixing their customer's problem.

2.5 Product Shutdown Phase

There are many reasons why an organization may stop work on a product:

- a. The product may have been a success in the past, but now has reached the end of its useful life,
- b. The product was successfully completed, but now is being turned over to the customer or another organization. This happens frequently with government or foreign contracts,
- c. The product may have been a failure for business, management or technical reasons.

Whichever the case, having kept product source under **SABLE** throughout the life cycle can aid the transition. In all cases, not just the last, a wrap-up analysis should be done as part of a continuous improvement program. The wealth of quality and productivity data **SABLE** has accumulated can be invaluable to this analysis. For example, **SABLE** stores every MR that has passed through it in its inactive data base. In the first case above, even though this product has outlived its usefulness, the organization may be developing a sequel or, at least, their next product may be for the same business sector as or have characteristics similar to the previous product. 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 serving as examples for upcoming work.

We have all seen how much longer the effective transfer of knowledge takes than we had initially thought. The sending side always seems to have trouble gathering up all the relevant, up-to-date information and the receiving side seems to take longer to pick up the information than the sending side expects because the receiving side is not steeped in, and usually does not have access to, the product's history the way the sending side is. Because of the orderliness that **SABLE** supports and its ability to store (historical) documents and MR history, **SABLE** can make it easier for both sides in the transition. It should be noted that for the same reasons, the use of **SABLE** can make it easier for new members to get on board ongoing projects.

Depending on the reason(s) for a product's failure, a more in-depth analysis may be called for. Here again, the quality and productivity data that **SABLE** provides can support such an analysis. If the project was progressing well but business reasons caused a halt, then this case may be like case one above. If project is on hold for some reason, then this case may be like case two above.

3. SABLE and Project Management

Above, we have looked at product life cycle phases, which wax and wane as time goes on. There are, however, project management needs that remain more or less constant throughout the life cycle and in this section we look at how **SABLE** supports these. **SABLE** is not a substitute for good project management, it is a that helps managers do a good job.

3.1 Organizational Efficiency

The application and coordination of resources is one of a manager's major concerns. For an organization to function efficiently, large tasks must be broken down and assigned and responsibilities must be made clear. Priorities and due dates must be set. **SABLE** provides managers with the following capabilities:

- a. Before committing resources to do a task, a manager may want to know more about what is involved. This may happen when the task is complicated, open-ended or of questionable value. In these cases, the manager may assign someone to study the problem and to propose a solution by a certain date. **SABLE** provides managers with a formal method for accomplishing this.
- b. When an MR is accepted to be worked, it is assigned to a person or a team and a priority and a due date may be set. As mentioned above, the people who have been assigned the task are notified. If necessary, an MR may be reassigned or even unassigned.
- c. If the task described in the MR is large and/or it spans several areas of responsibility, it may be subdivided into other MRs (called spawned MRs) and these may each be assigned to the appropriate people. For example, a new feature in a software product may require a change to the human/machine interface, an application program, the data base and the user guide. In addition, each of these areas may be the responsibility of a different team. Using spawned MRs allows this task to be broken down into smaller tasks that can be assigned, tracked and progress separately.
- d. **SABLE** provides a grouping capability that allows MRs to be grouped and named, say, for a maintenance release, and staff to be grouped into teams, say, the integration test team. These groupings are not only a shorthand convenience, but they also define the contents of feature packages and the organization of staff into teams and responsibilities of those teams.

3.2 Communication

One of the key reasons that projects run into trouble is the lack of adequate communication among team members. We have already mentioned **SABLE**'s support for documentation (formal communication) and the automatic communication to appropriate team members when an MR changes state, but there are several other ways that **SABLE** supports intra/inter-project communication:

- a. **SABLE** provides the kind of standard reports that are required in a well-managed project. In addition, **SABLE** allows the users to customize reports and provides a data extract file so user can roll their own special-purpose reports as needed.
- b. Bigger projects usually work on more than one machine. For example, development may be done on one machine and system test may be done on another. **SABLE** operates in a multi-machine (a host and satellites) environment where the local area network can be either Datakit™ or Ethernet™/3B Net™ and it also works with Remote File Sharing.
- c. Often different organizational entities will co-operate on an effort. 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 a joint sale. In these cases, it is important for the entities to transmit MR information (both MR descriptions and MR status) back and forth. For example, the development organization may encounter a situation that requires a change to both the product and the

™ Datakit is a trademark of ATT-BL. Ethernet is a trademark of Xerox. 3B Net is a trademark of ATT-BL.

requirements and, thus, would want to forward the MR to the systems engineering organization. **SABLE** supports these types of working arrangements because MR information may be sent from one **SABLE** instance to another.

3.3 Quality Assurance

In addition, to its support of the product testing phase discussed above, **SABLE** by its very nature is a quality assurance tool. MR tracking is one of key features of **SABLE** and a trouble report is one type of MR. Because of the records that **SABLE** keeps as MRs pass through it, **SABLE** can supply the Quality Assurance team with a wealth of status and history information. In particular, **SABLE** provides the quality metrics, such as cumulative fault density, mean time to close and average number of open troubles, as defined and requested by the ATT-BL Quality Assurance Center.

3.4 Access Control

For software, firmware and documentation projects, the source *is* the product and must be protected against unwanted or unofficial changes. **SABLE** provides the following access control mechanisms:

- a. In order to execute **SABLE** commands the user must have a **SABLE** login. The **SABLE** administrator is to only one who can add these logins, which are stored in **SABLE**'s Personnel Tracking System. Once a login has been entered, the user can update his or her own administrative data such as location and room, phone and organization numbers. This restricts access to authorized users only.
- b. In order for a user to modify source under **SABLE**, he or she must not only give a valid MR Id, but that MR must be assigned to him or her or to a team of which he or she is a member. This helps make sure that source is modified for the right reasons.
- c. Projects have the option of recording the history of all **SABLE** activities. This history serves as an audit trail and tracks who changed what when.

These mechanisms form a second line of defense and, obviously, depend upon appropriate security precautions being in place on the machine on which **SABLE** is running.

4. User Considerations

SABLE has provided many features to increase ease of use and to allow users to tailor it to their needs.

4.1 The Human/Machine Interface

SABLE provides both a full-screen and a command line human/machine interface. The full-screen interface does *not* require a dot matrix display terminal. The full-screen interface has pop-up menus, help messages and error messages. The delay for the pop-up menu may be set by the user, so novice users can get the menus sooner and experienced users, who usually don't need them, will only get them when they are stumped. Figure 1 shows a typical **SABLE** screen, which is used to create an MR. Figure 2 shows the same screen partially filled in. The user waited to see which were the legal values for the **System** field and the menu for that field has popped up. The help messages pop up on demand and the error messages pop up as errors occur. Once a menu has popped up, the system uses a first match algorithm to select a choice given what the user has typed so far. This speeds data entry in these cases. The interface gives users the standard field navigation capabilities as well as some other, more advanced capabilities. For example, the user may make multiple selections from a pop-up menu, not just the usual one. 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 the necessary data may be entered, but the screens are reasonably designed and ease to use for the more typical input.

SABLE provides a line command interface for two reasons. On rare occasions, a user may have to access **SABLE** from a terminal without a screen display. More typically though, a user may want to writeshells that call **SABLE** commands. This gives users added value by allowing them to create special-purpose, project-specific commands or to roll their own reports, for example. Where not limited by inherent differences in the media, for example, field navigation, the line command interface and the full-screen interface have the same behavior.

line right 3.5; arc cw; line down 2.3;arc cw; line left 3.5; arc cw line up 2.3;arc cw

logid:userid SABLE 2.0 Software Administration System 01/23/88
effid:sable MR Management Command Functions 12:21:27

Create a New Modification Request

Originator PTS ID: _____ Origination Date: _____
Request Severity: __ Required Date: _____

Product: _____
System: _____
Subsystem: _____
Release: _____
Site: _____

Category: _____

Abstract of Request: _____
Request Desc File: _____

Figure 1: Example of a SABLE Screen

line right 3.5; arc cw; line down 2.5;arc cw; line left 3.5; arc cw line up 2.5;arc cw

logid:userid SABLE 2.0 Software Administration System 01/23/88
effid:sable MR Management Command Functions 12:23:27

Create a New Modification Request

Originator PTS ID: kaiser!steve_ Origination Date: 01/23/88
Request Severity: 3 Required Date: _____

Product: sable2.0 _____
System: _____
Subsystem: _____
Release: _____
Site: _____

admin
mrmgmt
srcmgmt

Category: _____

Abstract of Request: _____

Request Desc File: _____

Figure 2: Example of a Pop-up Menu with Project-Specified Values

4.2 User Programmability

SABLE gives each user some flexibility in tailoring his or her interaction with it. Similar to the user profile in UNIX™, the **SABLE** user can set personal interface choices in his or her Personnel Tracking System (PTS) record. The user can select:

- a. The type of human/machine interface, full-screen or line command, the user would like,
- b. The editor he or she wants to use when an editor is called for. For example, when an MR description or resolution is entered,
- c. The time delay before a menu pops up. Expert users usually set this delay longer because they don't need the help and novice users set this shorter because prompting is needed more often,
- d. Whether the user wants verbose or terse help and error messages. The expert users usually opted for the terse messages, while the novice users start out choosing the verbose messages,
- e. On which machine the user wants to receive mail from **SABLE**. The user can also set other administrative/organizational information, such as phone, room and department numbers.

Once a user has been added to **SABLE**, he or she always has immediate access to the above data so it can be changed on the fly, if necessary.

™ UNIX is a trademark of ATT-BL.

4.3 Project Programmability

Given that each project has slightly different needs, either because of historical reasons or current managerial or contractual reasons, **SABLE** provides some flexibility at the project level as well as at the user level. Project-wide choices can be made concerning:

- a. Which fields, of those not required by **SABLE**, will appear on screens and which of these fields are mandatory,
- b. The legal values that appear in many of the pop-up menus. In fact, the user can set up a menu tree such that once a value has been selected for a prior field, only certain values are acceptable for subsequent fields. For example, in Figure 1, once a project-specified value has been selected for **System**, only certain values are legal for **Subsystem** and the menu tree can be set up so only those values appear on the menu for **Subsystem**.
- c. Whether or not a project-specific MR description template will be used to collect additional and/or format information,
- d. Whether or not a complete history of **SABLE** activities is kept,
- e. Which of the optional test states will be used,
- f. Who will have permission to execute which commands,
- g. Whether or not project-specific reports will be provided,
- h. The design of the project directory structures,
- i. Since **SABLE** runs on all the UNIX machines in common use throughout AT&T, projects have wide flexibility in where they run it.

4.4 User Documentation

Thorough, professionally published user documentation is provided with **SABLE**. There is a user's guide[4] and an administrator's guide[5]. Each is rich in examples as well as describing the commands and process. Each is augmented by a quick-reference card and each is targeted a specific portion of the user community. The user guide is structured such that the user need only read the introductory material and the sections for the commands he or she will be using. In addition to this documentation and the on-line help provided with **SABLE**, initial training and ongoing consultation is also provided.

5. Architecture

Figure 3 gives an overview of key components of the **SABLE** architecture and ties together much of what has been discussed above. Starting from the top left and sweeping down to the bottom right, a synopsis of each component is:

- a. User Interface/Screen Manager - Provides the full-screen and line command human/machine interface. Table-driven techniques are used and users may make certain project-specific changes to the interface.
- b. Communications - Handles **SABLE** to **SABLE** communication, **SABLE** running in a multi-machine environment and communications with other systems such as a regression test system or other sources of MRs.
- c. PTS - The Personnel Tracking System handles **SABLE** logins and administrative and profile data of each of the users.

- d. MR Management - Provides the Modification Request tracking and reporting function.
- e. Source Management - Provides the source change tracking and control function. The Source Code Control System (SCCS) is used to provide part of this functionality.
- f. Date Bases:
 - i. The Active MR Data Base stores information about open MRs and the source that has been changed because of them.
 - ii. The Inactive MR Data Base is identical to the Active Data Base and stores information about MRs that have closed. This data is useful for post-project analysis. Separating open and closed MRs speeds access to the open MRs.
 - iii. The Source Data Base is where product source is actually stored and is managed by SCCS.
- g. Directory Structures - **SABLE** populates these via its GETVERSION command. The two shown here could be, for example, one (partially populated) used by a developer for recent changes and the other (fully populated) could be the official source directory structure. **NMAKE**'s viewpathing capability could be 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. Only two directory structures are shown here, more may be used when required. As discussed below in the Manufacturing section, **NMAKE** and **SABLE** will work together more closely in the future.
- h. Software Quality Assurance Reports (SQAR) - The data for these reports is collected from **SABLE**'s data bases and formatted such that it can be used to produce the reports locally or sent to the ATT-BL Quality Assurance Center if the project participates in SQAR.

6. Future Directions

Although **SABLE** is already a robust product administration system, because of its place in the development process, its (along with **NMAKE**'s) widespread acceptance* and the ever increasing demands placed on the AT&T development community, there is an ongoing research and development program for **SABLE** and **NMAKE**. The sections below discuss what is on the horizon for these tools.

6.1 Non-ASCII Text

Currently, the standard source control system used in **SABLE** can only store ASCII text. There are several instances when projects would like to be able to control non-ASCII text. For example, there is a need to control object code as well as source code when doing incremental compiles and there is a need to control digitized data for pictures and the like too. A future release of **SABLE** will provide this capability as well as loosening some size limitations and improving the speed of the source control system.

6.2 Manufacturing

Today there is a handshake interface between **SABLE** and **NMAKE**. **SABLE**'s GETVERSION command populates directory structures with the desired versions of the product and **NMAKE** builds the product by viewpathing through the directory structures in the user-specified order. Both are powerful, independent tools as they stand, but their synergy can be even more powerful. For example, instead of asking **SABLE** to retrieve a

* **SABLE** is in use or being evaluated by over 100 projects throughout AT&T and, similarly, **NMAKE** is in almost 200 projects.

certain version of the product and then asking **NMAKE** to build it, the user should be able to ask that a certain version of the product be built and have **SABLE** and **NMAKE** work together to honor the request.

This would not only be a user convenience, but would allow much more. For example, today each time a **GETVERSION** command is done, **SABLE** has no idea if a version of the product will actually be built from that source or if the source will be modified outside of **SABLE** before it is built. Similarly, **NMAKE** does not know whether the source it is working with came from **SABLE** or not. The upshoot of all this is that despite good intentions one can never be sure that what left **SABLE** was actually used to build the product. This has implications in deliveries to both system test and the field. With the manufacturing capability, not only will there be certainty about what is being built, but good records can be automatically be kept concerning when the product was built, what went into the product and who requested the build.

In addition **SABLE** and **NMAKE** can work more efficiently together than they can apart. For example, **NMAKE** determines what has to be rebuilt by determining what has changed and for safety sake, it rebuilds when there is a doubt. On the other hand, **SABLE** keeps track of what has changed and with the ability to control object code (discussed above) it will be able to give **NMAKE** more accurate change information. This arrangement will also be able to reduce the time stamp fuzziness involved in building a product on one machine with source from another machine.

6.3 Documentation

The documentation feature set provided by **SABLE** will continue to expand. Since we see **SABLE** as the project document repository, we want to add document classification and search capabilities so documents can be found quickly and new team members will have easy access to relevant project documentation. Authoring tools, electronic document ordering and distribution and hypertext seem fruitful and the extent of our work in these areas will be determined by our own investigations and through interactions with our current and potential customers.

6.4 More Support For Hardware Projects

SABLE has its roots in software administration, but our 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. Our document templating capability works as well for hardware-specific documents as any others and we will investigate what standard hardware-specific document templates we should be providing with **SABLE**. The ability to control non-ASCII text, discussed above, may be of value in storing digitized data and schematics. We will be working with hardware projects to determine the value of these and other hardware-related features.

6.5 Cost of Quality and In-Process Quality Metrics

Because **SABLE** collects additional information each time an MR changes state, it is in perfect position to gather data about how long a change is scheduled to take and how long it actually took. Getting a handle on the reasons for and the amount of rework done on his or her project can give a manager additional insight into how the development process can be improved. Similarly, the collection of in-process metrics, like inspection results, can help a manager make mid-course corrections. Since **SABLE** already keeps MR data, it is natural for **SABLE** to collect in-process quality metric data or, at least, cooperate with a system that does. We will be investigating the customer needs and options here with the goal of providing our users with an integrated tool environment.

AT&T – PROPRIETARY

Use pursuant to Company Instructions.

6.6 Asset Management

The reuse of software and even specifications and designs can dramatically increase a project's productivity and quality. We consider these reusable items to be project and even company assets. Since **SABLE** is the repository for these assets and because reuse is so important, it is natural to extend **SABLE** in the area of asset management. One extension would be to allow for the classification, description and search of assets, similar to what was described for documents alone above. This way, before writing something from scratch, the asset pool could be checked to see if something already exists that does the job or is at least close. In particular, this can save new people on larger projects considerable time.

Another extension would be to allow source files as well as MRs to be sent from one **SABLE** instance to another. This mechanism would simplify reuse across projects, as discussed in the Project Shutdown section above, because consistent, complete packages can be shared and because the MR history of the asset can be sent along with it to give the receiving organization a better basis on which to decide to reuse.

6.7 Supporting More Of The Development Environment

There is a whole body of useful or potentially useful tools that support what might be called "source analysis". These range far and wide, some examples are:

- a. Software module complexity metrics such as McCabe and Halstead,
- b. Software structural complexity metrics and the related software structure graphical displays,
- c. Document analysis such as provided by Writer's Workbench and beyond,
- d. Software information abstractors that support data dictionaries, browsers and the like,
- e. Consistency and standards checkers.

Since **SABLE** stores a product's source, it is in a perfect position to provide and/or support these types of analyses. **SABLE** brings many advantages over using the analyzers alone:

- a. As mentioned above, **SABLE** has all the source with which to work,
- b. Because of its ability to recreate different versions of the source for analysis, A-B comparisons of the characteristics of interest can easily be done. For example, a project can see the impact of some new practice or technology they've put into place,
- c. Because it has the MR data as well, it would be easier for a project to determine which characteristics affect its trouble rate and how,
- d. Since **SABLE** can act as a gatekeeper if a project so desires, it can run selected analyses as source is being stored in it,
- e. Because all project source can be stored under **SABLE**, it would be in a better position to support analyses of how characteristics of the input one phase affects its output. For example, how characteristics of the requirements documents affect the resulting software or comparisons between the software to be tested and the test scripts designed to test it,
- f. Since **SABLE** tracks source changes, keeping the analysis data up-to-date would be cheaper incrementally and it could also give a better indication about what has changed since the last analysis was done.

NMAKE with, perhaps, centrally supplied rules would be the natural engine to run the analyses. In fact, the type of synergy discussed in the Manufacturing section above would apply here too.

Standardization and support for these analyzers are obviously two areas of concern and, with input from our customers, **SABLE** could provide a standard set. On the other hand, there will probably be some special purpose needs and in these cases, it would make sense for **SABLE** to trigger user-supplied analysis programs.

7. References

- [1] System Development Framework
- [2] AT&T Technical Journal Article on PM (See Glenn S.)
- [3] *Quality by Design*, Issue 1 AT&T Bell Laboratories Quality Assurance Center
- [4] *SABLE User's Manual*, K. Flanagan
- [5] *SABLE Administrator's Manual*, K. Flanagan